PROYECTO DE SISTEMAS INFORMÁTICOS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Describiendo e implementando normas para sistemas multi-agente

AUTORES

Manuel Félix Díaz Jara

Inmaculada Magro García

Jorge Luis Queipo Bravo

PROFESOR DIRECTOR

Viviane Torres da Silva

[curso 2007/2008]

Abstract

Los sistemas multi-agentes compuestos por agentes heterogéneos, autónomos e independientes están normalmente gobernados por un conjunto de normas. Las normas establecidas regulan el comportamiento de los agentes indicando sus permisos, prohibiciones y obligaciones.

Este proyecto está dividido en tres etapas. En la primera etapa se implementa una aplicación de ayuda al usuario para la descripción de normas para sistemas multi-agente.

La segunda etapa del proyecto está relacionada con la implementación de las normas. Implementamos un transformador automático que recibirá la norma (resultado de la utilización de la aplicación anterior) y generará las reglas en Jess que activarán las normas, desactivarán las mismas e informarán a los agentes sobre las violaciones de estas normas.

En la tercera etapa del proyecto se implementara una aplicación multi-agente que utilizará la aplicación Jess. Los agentes consultarán la base de conocimiento Jess para saber cuáles son las normas activas, desactivas y violadas. Mostramos a través de un ejemplo la ejecución de las tres etapas del proyecto.

The multi-agents systems composed of heterogeneous, autonomous and independent agents are usually governed by a set of norms. The established norms regulate the behavior of the agents indicating their permissions, prohibitions and obligations.

This project is divided in three stages. In the first stage an application to help users while describing their norms is implemented.

The second stage of the project is related to the implementation of the norms. We hace implemented an automatic transformer that receives the norm (result of the use of the previous application) and generates the rules in Jess that will activate the norms, deactivate these norms and give information about the violations of these norms.

In the third stage of the project a multi-agent application is implemented to use the Jess application. The agents will consult the Jess base of knowledge to know which norms are active, inactive and violated. The three stages of this project are shown through an example.

Keywords: sistema multi-agentes, diagramas, modelado, normas, reglas.

ÍNDICE

1. Introducción

La administración de sistemas multi-agentes (MAS) copa con la heterogeneidad, autonomía y diversidad de intereses entre agentes que pueden trabajar con similares o diferentes objetivos. Varios sistemas de normas fueron propuestos para regular el comportamiento de los agentes definiendo qué acciones tienen prohibidas, permitidas u obligadas.

El enfoque de este trabajo está en la descripción de las normas y su implementación utilizando un sistema de reglas de inferencia llamado Jess. Este trabajo presenta una aplicación para auxiliar el usuario a describir las normas de su sistema y otra aplicación (integrada a la primera) capaz de general el conjunto de reglas en Jess que podrá ser utilizado para gobernar el comportamiento de los agentes de acuerdo con la norma. El mecanismo activa las normas y dispara las violaciones de acuerdo con la información recibida sobre la ejecución de las acciones.

Para describir las normas, hemos utilizado un lenguaje normativo descrito en BNF. Este lenguaje posibilita la especificación tanto de normas dialógicas como de normas no dialógicas. Las normas dialógicas regulan la interacción entre los agentes y las normas no dialógicas regulan las tareas ejecutas por los agentes, por ejemplo, el acceso a recursos, su compromiso para jugar un role o su movimiento entre entornos y organizaciones.

Tanto las normas dialógicas como las no dialógicas regulan el comportamiento de los agentes estableciendo restricciones en la ejecución de acciones dialógicas y no dialógicas, respectivamente. Los dos tipos de normas son capaces de identificar obligaciones, permisos y prohibiciones, condiciones temporales para la ejecución de las normas y sanciones (recompensas y penalizaciones). Las sanciones no sólo se refieren al conjunto de acciones a ser ejecutado para penalizar a un agente, si no también acciones a ser ejecutadas para recompensar al agente.

La aplicación de descripción de las normas ayudará al usuario a escribir las diferentes reglas que él necesitará para su aplicación MAS. La aplicación le ayudará indicándole cuales son las diferentes opciones que tiene que seguir cada parte de la regla, además de indicarle las partes que le faltan por especificar. También le permite la eliminación o la modificación de las partes especificadas anteriormente.

Después de haber especificado una norma el usuario puede utilizar la aplicación de generación de reglas para generar automáticamente el conjunto de reglas en Jess capaz de gobernar el comportamiento de los agentes de acuerdo con la norma. La entrada para el generador es la norma descrita con el lenguaje normativa y la salida es el conjunto de reglas en Jess.

La principal ventaja de usar Jess es la posibilidad de cambiar dinámicamente el conjunto de reglas definidas en Jess durante la ejecución de la aplicación. Esto es fundamental cuando no todas las normas serán definidas durante el diseño de la aplicación y las normas ya definidas van a ser modificadas durante la ejecución.

Este documento está organizado de la siguiente forma. En la sección 2 incluye toda la documentación de la aplicación que ayuda al usuario a describir las normas que gobiernan los MAS. La documentación sobre el sistema que implementa dichas normas en Jess se encuentra en la sección 3. En la sección 4 se incluye un ejemplo de un MAS que incluye normas implementadas con la aplicación de la sección anterior.

2. Aplicación para ayudar al usuario a describir las normas del sistema

2.1. Introducción

En esta primera parte del trabajo se ha diseñado un asistente que ayuda al usuario a crear de forma fácil e intuitiva una norma de un sistema multi-agente. Los objetivos de la aplicación son proporcionar al usuario la posibilidad de crear una norma de las siguientes formas:

- **Sencilla**: el programador será capaz de definir normas sin necesidad de tener conocimiento alguno del lenguaje normativo, eso es, de la gramática BNF siendo utilizada.

- **Segura**: Esta aplicación también reduce las posibilidades de que el usuario se equivoque al describir la norma, puesto que sólo permite la elección entre las diferentes opciones válidas disponibles en la gramática para cada etapa de la descripción de la norma.

- **Intuitiva**: Para dar mayor facilidad al usuario, se han incluido diversas funciones adicionales como el uso de colores diferentes para las ramas no expandidas del lenguaje.

- **Rápida**: El sistema ofrece al usuario la posibilidad de no tener que escribir cada palabra de la norma, si no que da a elegir entre los diferentes terminales que pueden componer una parte de la norma. Además, el sistema permite deshacer o cambiar cualquier rama previamente definida por el usuario sin tener que volver a empezar a crear la norma.

- **Robusta**: Este asistente permite, sin modificación alguna en la implementación, el cambio de la gramática BNF siendo utilizada para la descripción de la norma. Esto resulta bastante útil puesto que en un futuro podrán aparecer nuevas normas que pueden ser creadas con nuestra aplicación sin ningún esfuerzo adicional.

2.2. Gramática utilizada para la descripción de las normas

Uno de los aspectos en fase de investigación es el lenguaje normativo que define las acciones de los agentes. El camino que se sigue consiste en enriquecer un framework existente (Electronic Institutions para negociaciones entre agentes) para incrementar su expresividad y flexibilidad. A partir de las ideas de [3],[4] y [5], los autores de [2] extendieron los conceptos deónticos y ampliaron las obligaciones, permisos, prohibiciones, violaciones y sanciones de las acciones de diálogo. Por último, [1] introduce los conceptos de acciones que no están relacionadas con el diálogo entre agentes [5], condiciones y situaciones temporales definidas para acciones no dialógicas, y define castigos y recompensas para estas acciones u otras normas. La gramática utilizada en la aplicación contempla esta última versión, siendo la primera parte de la aplicación independiente de la versión de la gramática utilizada.

2.3. Diseño de la aplicación

Para presentar el diseño de la aplicación utilizamos modelos de casos de uso (sección 2.3.1), diagramas de clase (sección 2.3.2) y diagrama de secuencia (sección 2.3.3). En la sección 2.3.4 justificamos el lenguaje y las bibliotecas utilizadas en la implementación y la arquitectura adoptada.

2.3.1. Modelo de Casos de Uso:

Los casos de uso sirven para modelar las funcionalidades del sistema desde el punto de vista del usuario. En nuestro caso, la aplicación consta de 11 casos de uso diferentes que describimos a continuación:

1.CREAR UNA NUEVA NORMA	**General**
Descripción: Permite que el usuario inicie la creación de una nueva norma.	
Actores: Usuario, aplicación.	
Precondiciones: El usuario ha pulsado en la opción Nuevo	
Flujo Normal: 1. El usuario puede empezar a crear una nueva norma.	
Flujo Alternativo:	
Postcondiciones: 1. Se crea la nueva norma a partir del símbolo inicial	

2.EXPANDIR UN SÍMBOLO NO TERMINAL	**General**
Descripción: Permite expandir un símbolo no terminal a cualquiera de las posibles opciones.	
Actores: Usuario, aplicación.	
Precondiciones: El usuario ha pulsado en un símbolo no terminal	
Flujo Normal:	

1. Se abre una ventana de selección para el usuario. 2. El usuario elige la opción deseada. 3. Pulsa el botón aceptar de la ventana.
Flujo Alternativo:
Postcondiciones: 1. Se expande el símbolo no terminal en la opción elegida por el usuario
❖ **OBSERVACIÓN:** caso de uso diseñado en el diagrama de secuencia posterior

3.DAR UN VALOR PARA UNA VARIABLE DE LA GRAMÁTICA	**General**
Descripción: Se asigna un valor para una variable de la norma.	
Actores: Usuario, aplicación.	
Precondiciones: El usuario ha pulsado en una variable	
Flujo Normal: 1. Se abre una ventana de selección para el usuario. 2. El usuario escribe el valor deseado. 3. Pulsa el botón aceptar de la ventana.	
Flujo Alternativo: 1.a El valor escrito no es un valor válido para la variable: Se envía un mensaje de error. Se vuelve al paso 1.	
Postcondiciones: 1. La variable pulsada tiene el valor deseado en la norma	

4.ELIMINAR LA OPCIÓN ELEGIDA PARA UN SÍMBOLO NO TERMINAL	**General**
Descripción: Se elimina un nodo anteriormente expandido de un símbolo no terminal	
Actores: Usuario, aplicación.	
Precondiciones: El usuario ha pulsado la opción de eliminar en un símbolo no terminal	
Flujo Normal: 1. Se elimina la opción elegida para el símbolo no terminal.	
Flujo Alternativo:	
Postcondiciones: 1. El símbolo no terminal queda sin expandir	
❖ **OBSERVACIÓN:** caso de uso diseñado en el diagrama de secuencia posterior	

5.ELIMINAR LA OPCIÓN ELEGIDA PARA UNA VARIABLE	**General**
Descripción: Se elimina un valor anteriormente dado para una variable	
Actores: Usuario, aplicación.	
Precondiciones: El usuario ha pulsado la opción de eliminar en una variable	
Flujo Normal:	

1. Se elimina el valor elegido para la variable seleccionada
Flujo Alternativo:
Postcondiciones: 1. La variable queda sin valor

6.CAMBIAR LA OPCIÓN ELEGIDA PARA UN SÍMBOLO NO TERMINAL	**General**
Descripción: Se cambia un nodo anteriormente expandido de un símbolo no terminal	
Actores: Usuario, aplicación.	
Precondiciones: El usuario ha pulsado la opción de cambiar en un símbolo no terminal	
Flujo Normal: 1. Se elimina la opción elegida para el símbolo no terminal. 2. Se abre una ventana de selección para el usuario. 3. El usuario elige el valor deseado. 4. Pulsa el botón aceptar de la ventana.	
Flujo Alternativo: 4.a El valor escrito no es un valor válido para la variable: Se envía un mensaje de error. Se vuelve al paso 2.	
Postcondiciones: 1. El símbolo no terminal cambia de valor	
❖ **OBSERVACIÓN**: caso de uso diseñado en el diagrama de secuencia posterior	

7.CAMBIAR LA OPCIÓN ELEGIDA PARA UNA VARIABLE	**General**
Descripción: Se cambia un valor anteriormente dado para una variable	
Actores: Usuario, aplicación.	
Precondiciones: El usuario ha pulsado la opción de cambiar en una variable	
Flujo Normal: 1. Se elimina la opción elegida para la variable pulsada 2. Se abre una ventana de selección para el usuario. 3. El usuario escribe el valor deseado. 4. Pulsa el botón aceptar de la ventana.	
Flujo Alternativo: 4.a El valor escrito no es un valor válido para la variable: Se envía un mensaje de error. Se vuelve al paso 2.	
Postcondiciones: 1. La variable cambia de valor	

8.GUARDAR LA NORMA ACTUAL	**General**
Descripción: Se guarda la norma actual en un archivo especificado por el usuario	

Actores: Usuario, aplicación.
Precondiciones: El usuario ha pulsado en la opción de guardar
Flujo Normal: 1. Se abre una ventana de guardado para la norma 2. El usuario selecciona/escribe el archivo en el que guardar la norma. 3. El usuario pulsa el botón de guardar.
Flujo Alternativo:
Postcondiciones: 1. La norma queda guardada en el archivo especificado por el usuario
❖ **OBSERVACIÓN:** caso de uso diseñado en el diagrama de secuencia posterior

9.MOSTRAR LOS AUTORES DE LA APLICACIÓN	**General**
Descripción: Se muestran los autores de la aplicación	
Actores: Usuario, aplicación.	
Precondiciones: El usuario ha pulsado en la opción de Autores	
Flujo Normal: 1. Se muestra una ventana con los autores 2. El usuario presiona el botón de aceptar	
Flujo Alternativo:	
Postcondiciones:	

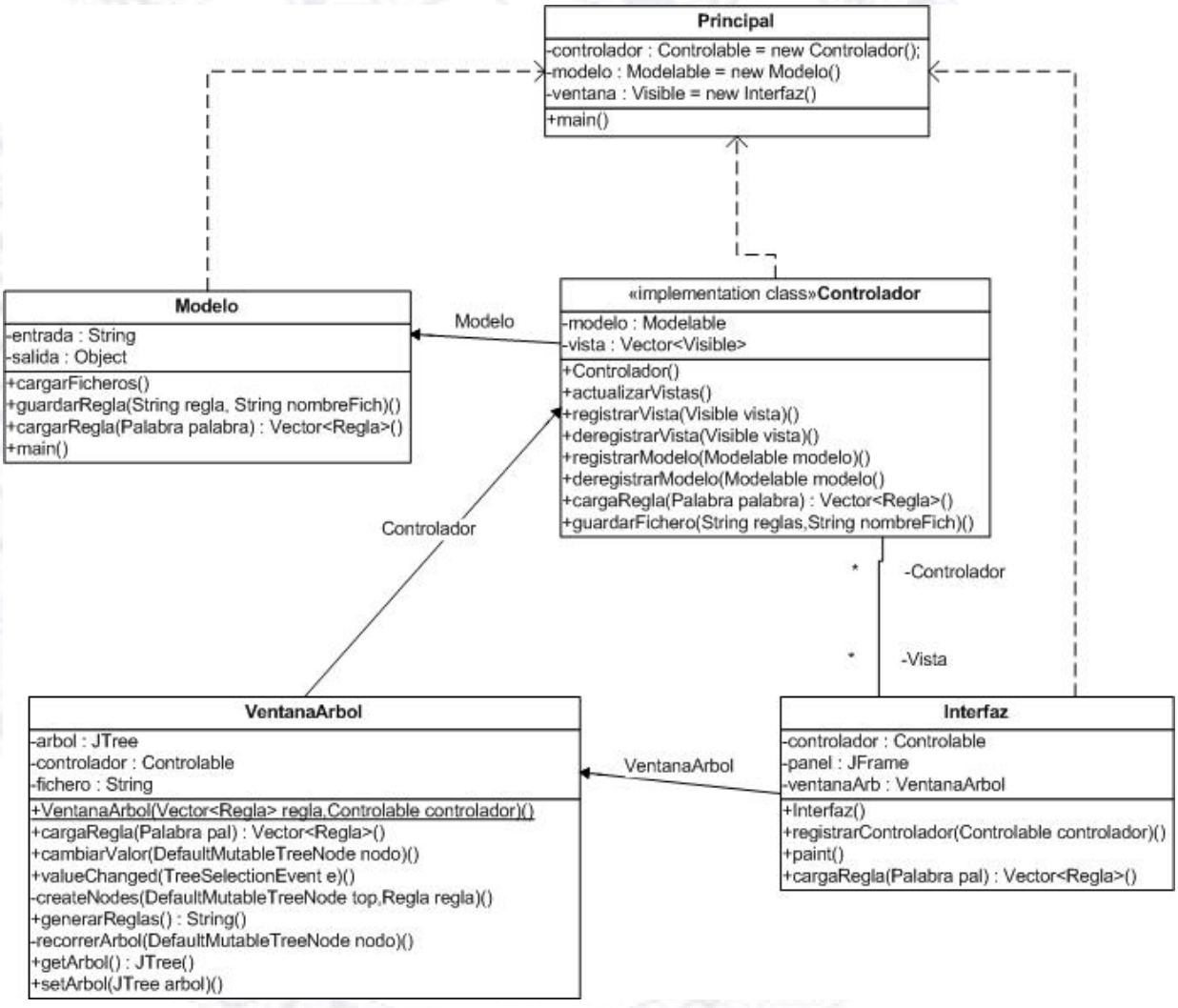
10.MOSTRAR EL MANUAL DE LA APLICACIÓN	**General**
Descripción: Se muestra el manual de la aplicación	
Actores: Usuario, aplicación.	
Precondiciones: El usuario ha pulsado en la opción de Manual	
Flujo Normal: 1. Se muestra una ventana del navegador con los autores	
Flujo Alternativo: 1.a No se puede tener acceso al manual: Aparece un mensaje de error	
Postcondiciones: 1. El manual queda a disposición del usuario para su lectura	

11.SALIR DE LA APLICACIÓN	**General**
Descripción: Se sale de la aplicación	
Actores: Usuario, aplicación.	
Precondiciones: El usuario ha pulsado en la opción de Salir o en la parte superior derecha de la aplicación	
Flujo Normal: 1. Se sale de la aplicación	

Flujo Alternativo:
Postcondiciones: 1. La aplicación queda cerrada

2.3.2. Diagrama UML o diagrama de clases

El diagrama de clases de UML describe la estructura del sistema mostrando sus clases, atributos y relaciones entre ellos. En este caso nosotros mostramos las cinco clases más importantes del diseño.



2.3.3. Diagramas de secuencia

El diagrama de secuencia se utiliza para modelar la interacción entre los objetos de un sistema. Debería haber un diagrama de secuencia para cada diagrama de casos de uso, pero en este documento sólo están diseñados diagramas de secuencia para los casos de uso más relevantes.

El diagrama de secuencia está compuesto por el nombre del caso de uso que representa, el usuario, el fichero de entrada, todas las clases con las que interactúa y las funciones de la implementación que utiliza. Cabe destacar que los diagramas de secuencia se refieren al diseño de la aplicación, por lo que se describen en el lenguaje Java en el que fue creada la aplicación.

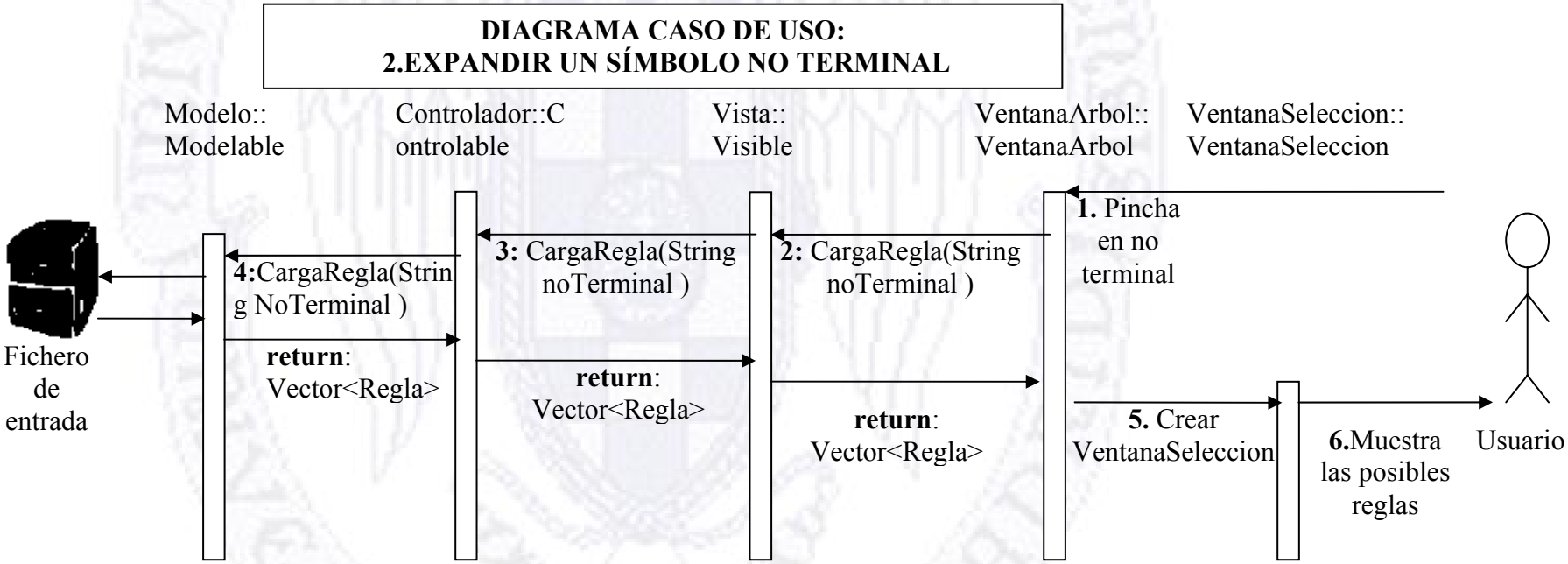


DIAGRAMA CASO DE USO:
8.GUARDAR LA NORMA ACTUAL

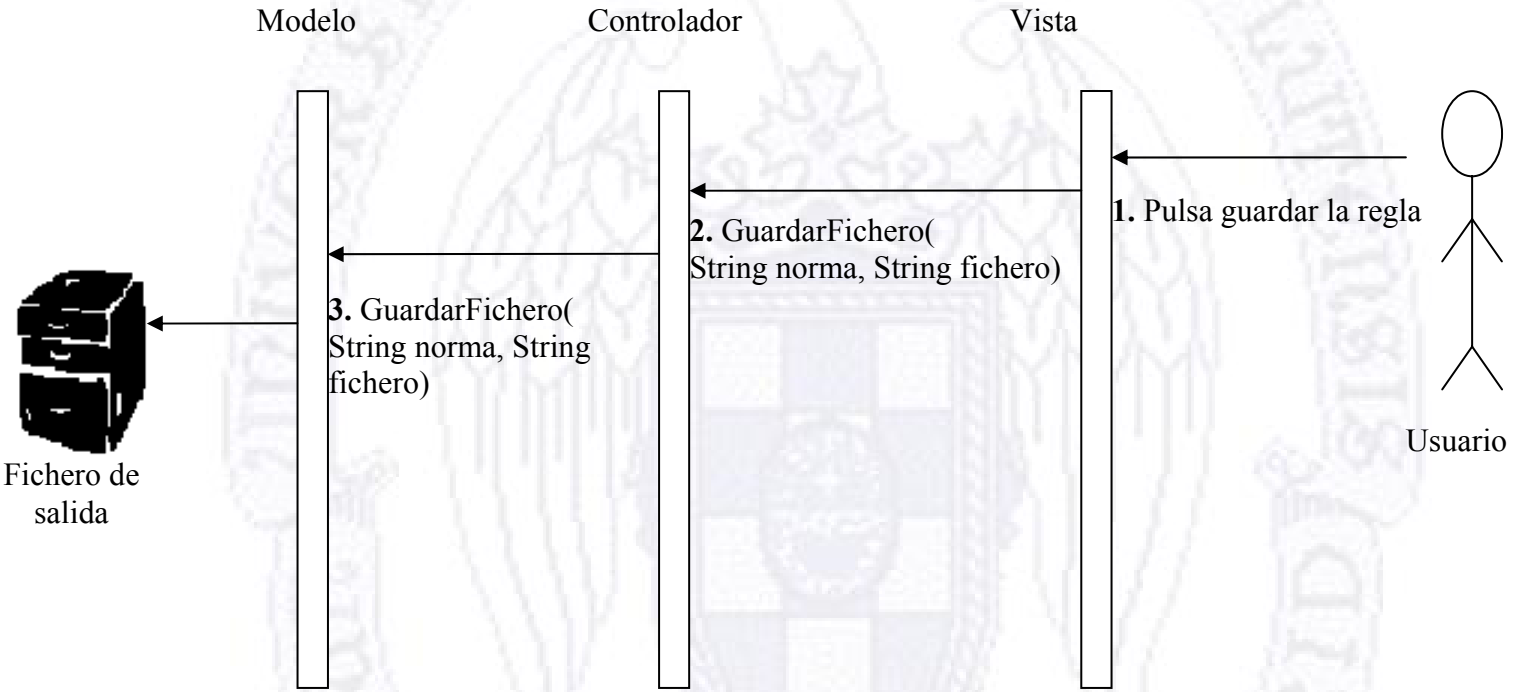


DIAGRAMA CASO DE USO:
4. ELIMINAR LA OPCIÓN ELEGIDA PARA UN SÍMBOLO NO TERMINAL

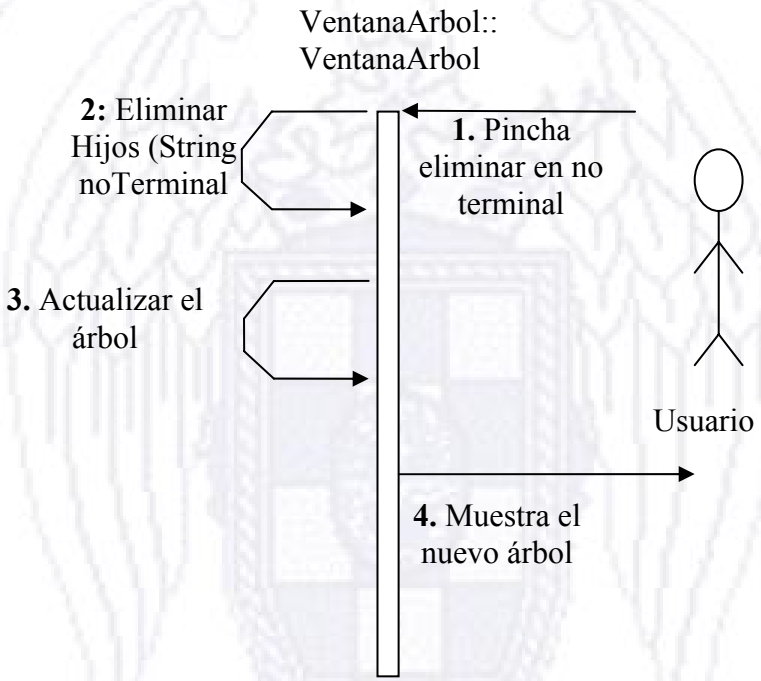
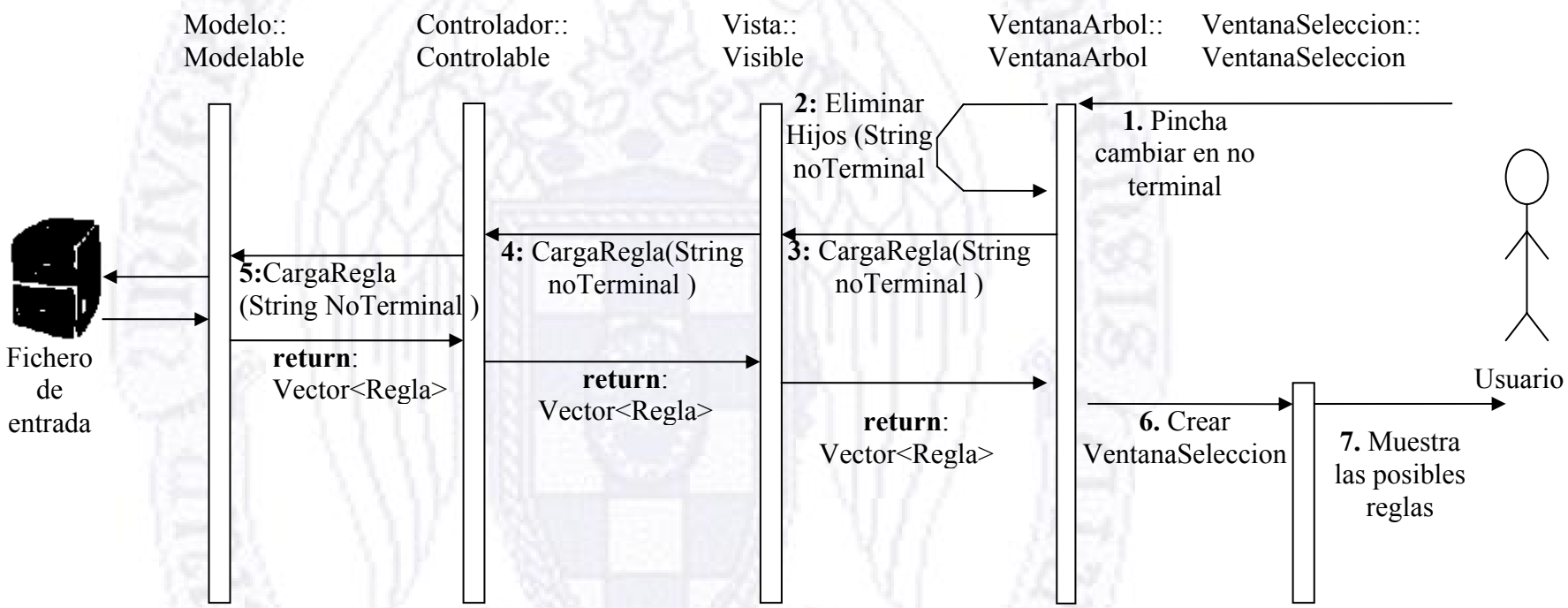


DIAGRAMA CASO DE USO:
6. CAMBIAR LA OPCIÓN ELEGIDA PARA UN SÍMBOLO NO TERMINAL



2.3.4. Especificación del programa

Lenguaje:
El lenguaje empleado para la creación de este programa es Java, ya que nos ofrece las siguientes características:

- Podemos utilizar la metodología de la programación orientada a objetos
- Permite la ejecución de un mismo programa en múltiples sistemas operativos
- Incluye por defecto soporte para trabajo en red.
- Ejecuta código en sistemas remotos de forma segura.
- Es fácil de usar y toma lo mejor de otros lenguajes orientados a objetos.

Arquitectura:
La arquitectura utilizada es la Modelo-Vista-Controlador en la que se distinguen tres partes:

- **Modelo**: gestiona los datos de la aplicación
- **Vista**: gestiona cómo se muestran los datos, es decir, la interfaz.
- **Controlador**: determina qué modificaciones hay que hacer cuando se interacciona con cada objeto.

¿Por qué elegir este tipo de arquitectura? Principalmente porque presenta varias ventajas frente a otros tipos de arquitecturas. La primera de ellas es que permite tener diferentes vistas para un mismo modelo. Proporciona un mecanismo de configuración a componentes complejos más tratable que el basado en eventos. Y además, puesto que la representación no era nada sencilla teniendo en cuenta la cantidad de datos que se iban a manejar, esta arquitectura nos permite construir nuevas vistas sin necesidad de modificar el modelo.

Interfaces gráficas de usuario (GUI):
Usamos bibliotecas de componentes tales como, Abstract Windowing Toolit (AWT) y Swing, Java Fundation Classes.

- La primera de ellas, AWT, es independiente de la plataforma con un nivel básico y experimental. AWT es una biblioteca de componentes gráficos, que permite utilizar los componentes nativos de cada sistema operativo.
- La siguiente también tiene una funcionalidad independiente de la plataforma a la cual se le pueden añadir nuevas funcionalidades. Permite emular la apariencia de los componentes nativos manteniendo la independencia de la plataforma.

Ambas bibliotecas son utilizadas para el diseño de la interfaz gráfica del sistema.

La estructura general de nuestro diseño está formada por:

- Un *JFrame* que formará el contenedor principal, al cual hemos añadido componentes, tales como contenedores intermedios del tipo *JPanel* y *JMenuBar*. El componente más importante que hemos utilizado ha sido el *JTree*, puesto que después de varios intentos fallidos con estructuración de ventanas, nos dimos cuenta que esa no era la mejor forma de organizar la interfaz, la elección final fue el nombrado *JTree* con el cual podemos crear una estructura de árbol en el que las hojas son nodos terminales (Identificadores) o bien no terminales que aún no fueron expandidos. Los nodos intermedios se corresponden con no terminales ya expandidos. Una forma de diferenciarlos son los

colores, que vienen explicados en la parte de manual, apéndice **¡Error! No se encuentra el origen de la referencia.**.

- Cada vez que se pulsa a un no terminal pueden aparecer dos tipos de ventana, la primera estaría compuesta por un *JComboBox* que implementa un cuadro combinado desplegable, en el que se agrupan las funcionalidades del no terminal pulsado. La segunda estaría formada por un *JTextField* la cual solicita el Identificador del no terminal al que pertenece. Esta distinción se realiza en función de dos parámetros, el primero es la identificación de los no terminales *Identifier, Numbers* y *Digit* con control de entrada; el segundo es más genérico y se basa en que si un no terminal empieza sin el símbolo "<" entonces pide automáticamente un *String* al usuario. La principal ventaja de implementarlo así es que el usuario no necesita ajustarse a un tipo de gramática fija para usar el programa.

3. Traductor automático de la descripción hasta la implementación

3.1. Introducción

La aplicación que hemos desarrollado es un traductor automático para generar el conjunto de reglas de Jess necesarias para gobernar una norma. El traductor ha sido desarrollado utilizando el lenguaje de programación Java.

La aplicación recibe como entrada la descripción de la norma en la gramática del lenguaje normativo. La salida de la aplicación consiste en la generación de un fichero .clp, formato usado por la aplicación Jess, con las reglas en Jess para gobernar la norma y los templates necesarios para la ejecución de las reglas generadas, como muestra la Figura 1. También se puede elegir añadir las reglas a un fichero .clp ya existente,

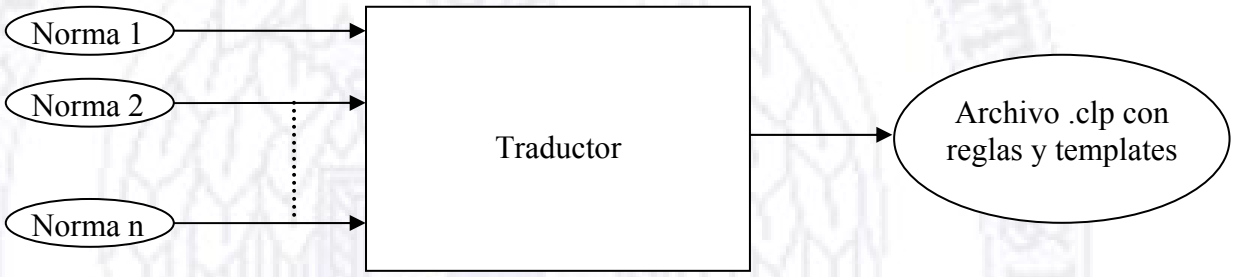


FIGURA 1: TRADUCTOR DE NORMAS A JESS

El traductor se puede usar para generar los diferentes tipos de normas que se pueden especificar mediante la gramática normativa. Los tipos de reglas que pueden ser generadas por el traductor son:

- **Obligaciones, permisos y prohibiciones simples.**
  Son las más sencillas puesto que no dependen de ninguna situación temporal o condición if.

- **Normas reguladas por acciones ejecutadas antes de la ocurrencia de un hecho.**
  Se pueden especificar obligaciones, prohibiciones o permisos para ejecutar una determinada acción antes de la ocurrencia de un hecho. Este hecho puede ser tanto una acción, una norma (si es violada o cumplida), una expresión o una expresión de tiempo, como un conjunto de ellas.

- **Normas reguladas por acciones ejecutadas después de la ocurrencia de un hecho.**
  Se pueden especificar obligaciones, prohibiciones o permisos para ejecutar una determinada acción después de la ocurrencia de un hecho. Este hecho puede ser tanto una acción, una norma (si es violada o cumplida), una expresión o una expresión de tiempo, como un conjunto de ellas.

- **Normas reguladas por acciones ejecutadas entre la ocurrencia de un hecho y la ocurrencia de otro.**

Se pueden especificar obligaciones, prohibiciones o permisos para ejecutar una determinada acción después de la ocurrencia de un hecho y antes de que se produzca otro. Estos hechos pueden ser tanto una acción, una norma (si es violada o cumplida), una expresión o una expresión de tiempo, como un conjunto de ellas.

- **Normas reguladas por acciones ejecutadas si ha ocurrido un hecho.** Se pueden especificar obligaciones, prohibiciones o permisos para ejecutar una determinada acción si ha ocurrido un hecho. Este hecho puede ser tanto una acción, una norma (si es violada o cumplida), una expresión o una expresión de tiempo, como un conjunto de ellas.

Los tipos de situaciones implementadas en el traductor que pueden producir la activación o desactivación de una norma son:

- **Acción ejecutada por un agente**: Una acción ejecutada por un determinado agente, ya sea una acción dialógica o no dialógica.
- **Expresión**: El valor de un determinado atributo de un agente.
- **Norma**: Consiste en la activación/desactivación de una norma debido a la activación, desactivación, cumplimiento o violación de otra norma.
- **Expresión de tiempo**: Consiste en la activación/desactivación de la norma si se cumple un determinado periodo de tiempo de una determinada acción o si es un día determinado.
- **Operación lógica de las diversas situaciones**: Consiste en la activación/desactivación de la norma si se cumple la conjunción de cualquiera de las anteriores situaciones(AND), la disyunción(OR), la disyunción exclusiva(XOR) o la disyunción negativa (NOR).

3.2. Jess

Jess (Java ExpertSystem Shell) es un lenguaje de programación basado en CLIPS. Fue creado por Ernest Friedman-Hill de Sandia National Laboratories en Livermore, CA. Es un lenguaje declarativo, es decir, basado en las matemáticas y en la lógica. Actúa en respuesta a entradas. Su funcionamiento está basado en reglas y mantiene una colección de hechos en su base de conocimiento. Se basa en encadenamiento hacia atrás (backwards chaining), es decir, parte de una hipótesis inicial y luego intenta demostrarla con la información que tiene.

Hay dos formas de utilizarlo como sistema experto o integrado en java:

- Como Sistema Experto (Sistema basado en conocimiento). Un sistema experto es un programa que emula al razonamiento humano, de manera que ejecuta ciertas reglas (definidas por el programador) sobre el conocimiento que se tiene. Este uso se aplica a los agentes inteligentes.
- JESS se ha escrito en Java y se puede integrar perfectamente a él accediendo directamente a sus clases y librerías para conseguir un desarrollo más rápido de las aplicaciones.

Los hechos son incluidos en la base de hechos y las reglas son activadas de acuerdo con estos hechos.

- Los hechos son descritos basados en plantillas que especifican la estructura del hecho.
- Emplea el algoritmo RETE (Algoritmo de Redundancia Temporal) que genera el conjunto de reglas aplicables a los hechos incluidos en la base de hechos.

Por lo tanto JESS puede dotar la aplicación de razonamiento, a partir de reglas que trabajan con la información inicial que se tiene. Así, poder crear agentes inteligentes que sean capaces de aprender y, de este modo, ser más eficientes.

3.3. Reglas de traducción

Para nuestro sistema, decidimos realizar una implementación en Jess para definir nuevos hechos (non-dialogical actions, rewards y punishments) y nuevas reglas (activación y desactivación de normas, cumplimientos, violaciones, castigos y recompensas).

Jess fue elegido por dos razones, la primera es que provee de interfaces para programar en Java y la segunda porque es posible hacer cambios dinámicos en el conjunto de reglas definidas en Jess durante la ejecución del programa Java. Por lo tanto lo usaremos para implementar el mecanismo que activa, desactiva las normas y lanza las violaciones o cumplimientos. Jess nos permite describir hechos y reglas que son lanzadas de acuerdo al estado de la base de hechos.

Para cada norma suele ser necesario describir cuatro reglas en Jess:

- Primera regla (rulei): se utiliza para indicar la norma condicionada por los hechos que la activan. Si los hechos están en la base de conocimiento, la regla estará lanzada y la norma activada.
- Segunda regla (ruleii): desactiva la norma. La activación de la norma está limitada y condicionada por algunos hechos de la base de conocimiento.
- Tercera regla (ruleiii): recompensa a los agentes si han cumplido la norma. En una obligación el agente será recompensado cuando ejecute la acción que está obligado, mientras la norma esté activa. En una prohibición, el agente será recompensado si la norma está desactivada y el agente no ha ejecutado la acción que estaba prohibida. En un permiso, no se aplican recompensas.
- Cuarta regla (ruleiv): se encarga de las violaciones y castigos. Una prohibición es violada si el hecho prohibido está en la base de conocimiento. Una obligación es violada si el hecho obligatorio no está en la base de conocimiento en el periodo correspondiente. Si una norma es violada, el castigo es aplicado cuando la violación es lanzada.

Ahora distinguiremos los distintos casos que nos podemos encontrar dependiendo de la norma que queramos definir.

3.3.1. Obligaciones, permisos y prohibiciones simples:

Son normas que se definen sin una situación temporal o condición, por lo tanto están siempre activas, es decir nunca serán desactivadas. Siendo así, la ruleii no es necesario definirla.

Para obligaciones y permisos sólo es necesario definir las reglas rulei y ruleiii que activan la norma e introducen las recompensas. Puesto que para ninguna de las dos es posible alcanzar el estado de violación, ya que en permisos la acción siempre podrá ser ejecutada y para la obligación puede cumplirse en cualquier momento.

Para prohibiciones no se define la ruleiii puesto que nunca acabará la prohibición de ejecutar una acción y por lo tanto nunca podrá ser recompensado. En este caso, son necesarias las reglas rulei para definir la activación de la prohibición y la ruleiv para definir el estado de violación y los castigos en el caso de que la acción prohibida esté en la base de conocimiento.

3.3.2. Normas reguladas por acciones ejecutadas antes de la ocurrencia de un hecho:

Obligación, permiso y prohibición de ejecutar la acción X antes de que ocurra el hecho W. Se comprueba si X ha sido ejecutado antes de que W ocurriera.

En el caso de la obligación y prohibición son necesarias las cuatro reglas, rulei para activar la obligación (prohibición); ruleii para desactivarla cuando W ocurra; ruleiii para establecer el estado de cumplimiento y recompensas, si W ocurre y X fue ejecutada, en el caso de la obligación o si X no está ejecutada, para el caso de la prohibición; ruleiv para la violación y para definir los castigos si W ocurre y X no fue ejecutada, en el caso de la obligación, o si X fue ejecutada, en el caso de la prohibición.

Para el permiso sólo son necesarias tres reglas, rulei para activar el permiso de ejecutar X; ruleii para desactivar el permiso si W ocurre; ruleiv para introducir las violaciones y castigos si W ocurre y X es ejecutado. No se puede verificar que la norma ha sido cumplida, por lo tanto no se puede definir la ruleiii.

3.3.3. Normas reguladas por acciones ejecutadas después de la ocurrencia de un hecho:

Obligación, permiso y prohibición de ejecutar la acción X después de que ocurra Y.

Para la obligación no es posible disparar la violación puesto que X puede ser ejecutada en cualquier momento después de que Y ocurra. Siendo así, la obligación debe ser reescrita usando el concepto de between.

El permiso, al igual que ocurre con la obligación, debe ser reescrito usando el concepto de between. Esto se debe a que tampoco es posible comprobar si el agente ha violado o ha cumplido la norma, puesto que X también puede ser ejecutada en cualquier momento después de que Y ocurra.

La prohibición la forman tres reglas, rulei que activa la prohibición si Y ocurre; ruleiii incluye el cumplimiento y recompensas si X es ejecutada antes de que Y ocurra (esta norma no es necesaria); ruleiv la violación y los castigos si X es ejecutada después de que Y haya ocurrido.

3.3.4. Normas reguladas por acciones ejecutadas entre la ocurrencia de un hecho y la ocurrencia de otro:

Obligación, permiso o prohibición de ejecutar la acción X después de que ocurra Y antes de ejecutar W. Para todos ellos será necesario definir las cuatro reglas.

Obligación: rulei activa la obligación de ejecutar X si Y ocurre; ruleii desactiva la obligación si está activada y W ocurre; ruleiii cumplimiento y recompensa si X se ejecutó y la obligación está activa; ruleiv violación y castigo si Y e W ocurren pero X no ha sido ejecutada.

Permiso: rulei activa el permiso de ejecutar X si Y ocurre; ruleii desactiva el permiso si está activa y W ocurre; ruleiii cumplimiento y recompensa si W ocurre; ruleiv violación y castigo si X es ejecutada pero Y no ha ocurrido todavía.

Prohibición: rulei activa la prohibición si Y ocurre; ruleii desactiva la prohibición si está activa y W ocurre; ruleiii dispara el cumplimiento e introduce las recompensas si Y e W ocurren y X no ha sido ejecutada; ruleiv introduce la violación y castigos si X es ejecutado cuando Y ha ocurrido pero W no, es decir cuando estaba activa.

3.4. Diseño de la aplicación

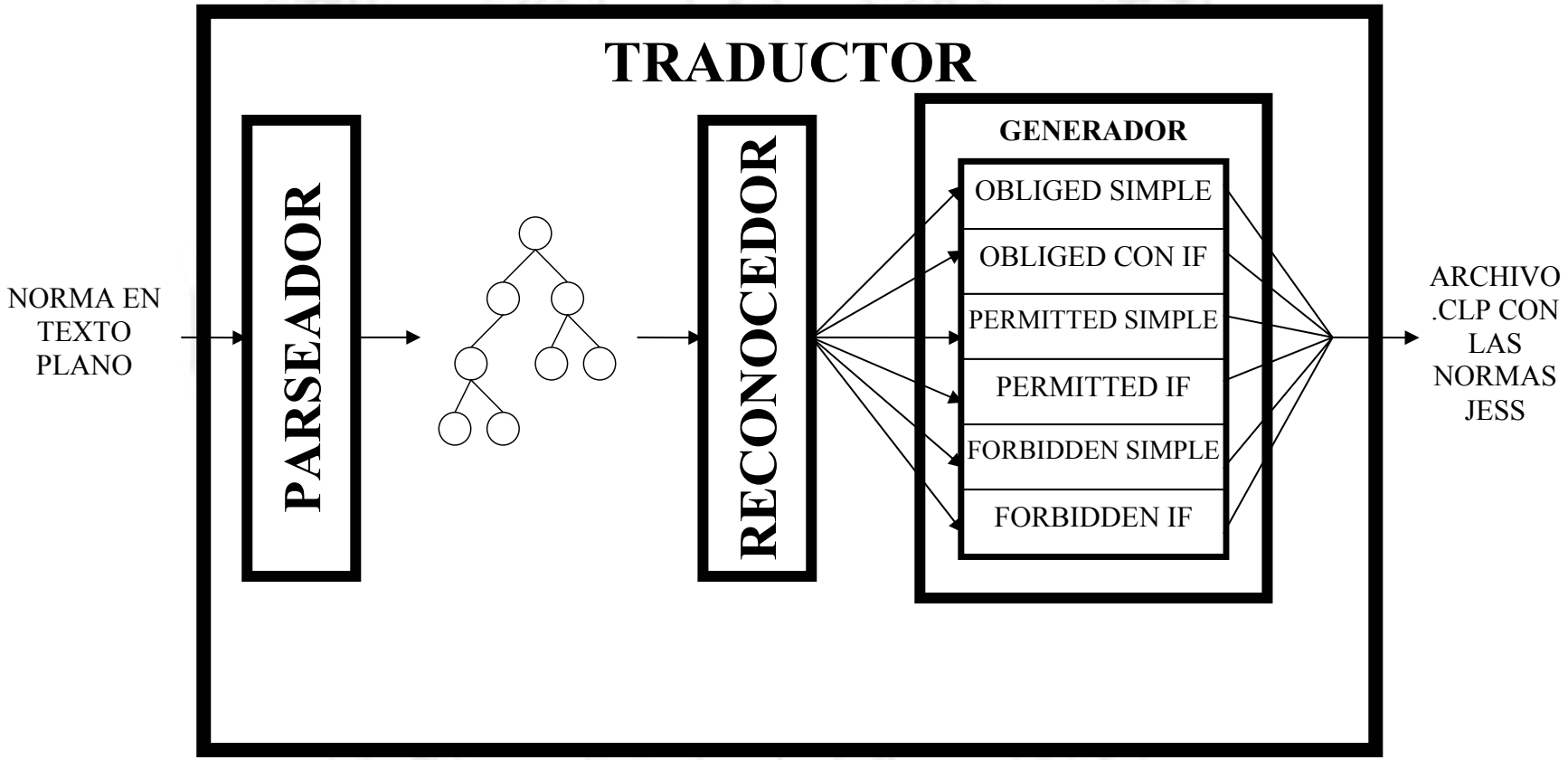


FIGURA 3: DISEÑO GENERAL DE LA APLICACIÓN

La aplicación está compuesta internamente por 3 módulos claramente diferenciados tal y como se ve en la figura 3. El primer módulo que interviene es el parseador, que recoge la norma especificada en la gramática normativa que utilizamos y genera el árbol correspondiente. Este árbol nos permite identificar las producciones que se han utilizado en la gramática para poder identificar las partes de la norma. Para generar el árbol utilizamos JavaCC, una aplicación realizada en Java de la que se explicará posteriormente su funcionamiento.

El segundo módulo es el reconocedor de normas y utiliza el árbol anteriormente generado. Este módulo es el encargado de identificar el tipo de norma para poder llamar al correspondiente sub-módulo del generador. Además, identifica la información necesaria que tiene que suministrar a dicho sub-módulo.

El tercer módulo es el encargado de generar las normas en el lenguaje Jess. Como cada norma se implementa de forma diferente según sea su tipo, el generador está compuesto por varios sub-módulos. El reconocedor de normas es el encargado de llamar al sub-módulo correspondiente y de proveerle de los datos que necesita para generar la norma en Jess. La salida de este módulo ya es un archivo .clp que incluye la norma implementada en lenguaje Jess.

3.4.1. Modelo de Casos de Uso:

Los casos de uso sirven para modelar las funcionalidades del sistema desde el punto de vista del usuario. En nuestro caso, la aplicación consta de un único caso de uso descrito a continuación:

1. Traducir normas	**General**
Descripción: Se recibe la norma en texto plano, y se hará todo el proceso para traducirla a formato .clp	
Actores: Usuario, aplicación.	
Precondiciones: El usuario ha pulsado en la opción de guardar	
Flujo Normal: 1. Se pasa el árbol para analizar a la clase Executer del paquete Global 2. Se interpretan y clasifican los nodos del árbol en la clase Executer 3. Generamos la regla en el paquete correspondiente Forbidden, Obliged o Permited 4. Devolvemos la representación de la norma al Executer 5. Executer añade todas las normas y se las devuelve a la clase Blend 6. La clase Blend devuelve el fichero de salida en formato .clp	
Flujo Alternativo:	
Postcondiciones: 6. La norma queda guardada en el archivo especificado por el usuario	
❖ **OBSERVACIÓN:** caso de uso diseñado en el diagrama de secuencia posterior	

3.4.2. Diagrama UML o diagrama de clases

El diagrama de clases de UML describe la estructura del sistema mostrando sus clases, atributos y relaciones entre ellos. En este caso, puesto que el diseño es tan

extenso hemos decidido mostrar la estructura y comunicación entre los paquetes del sistema.

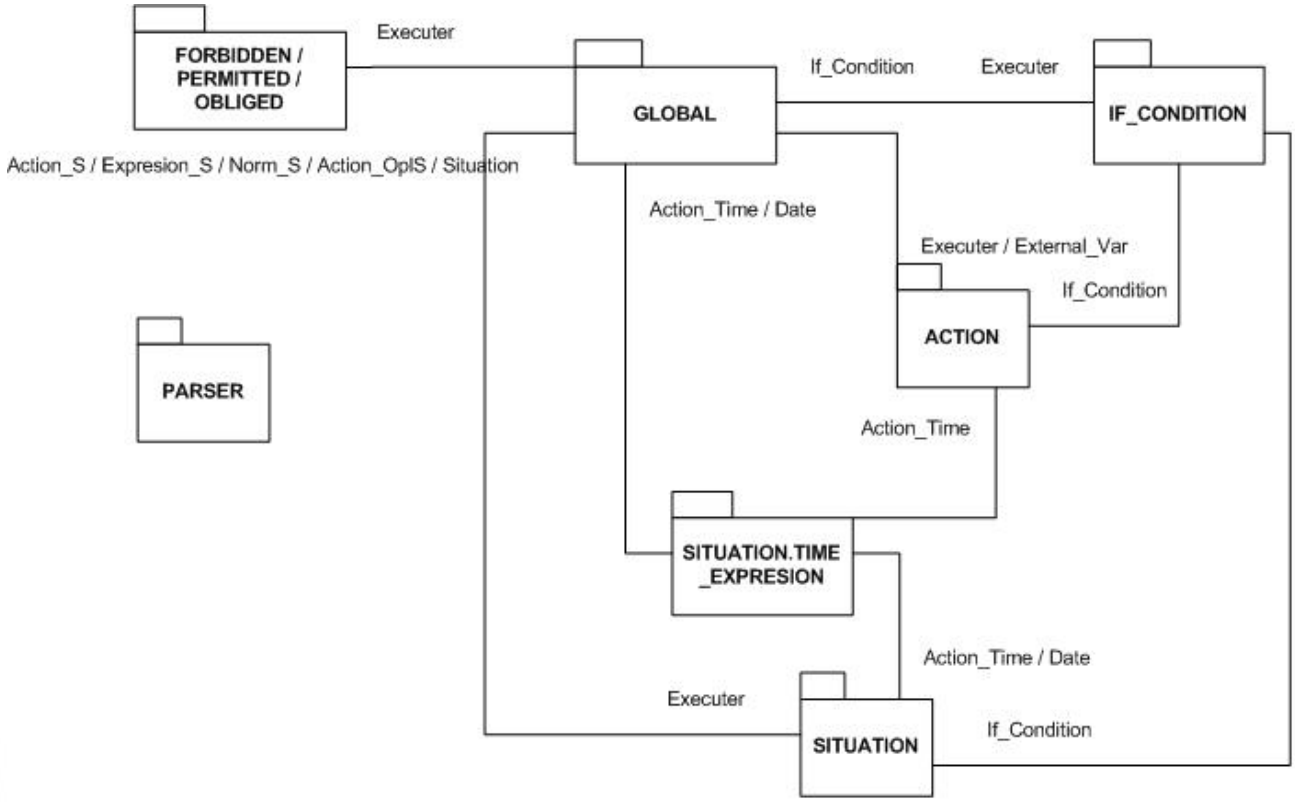


Figura 3.4.2.1 Diagrama de paquetes UML

Descripción de los paquetes de la figura 3.4.2.1:

Forbidden / Permitted / Obliged: son en realidad tres paquetes independientes pero están relacionados de la misma forma con el paquete global. En cada uno de ellos están definidas las reglas para cada una de las situaciones que hemos mencionado en otros apartados.

Global: está formado por Executer, ExternalVar, Presentation. La clase Executer es la encargada de interpretar el árbol de entrada del párser; ExternalVar se usa para añadir las variables y Presentation provee de formato de salida a las reglas.

If_Condition: como su propio nombre indica contiene la clase que define las condiciones IF que pueden aparecer en las normas.

Parser: está compuesto por las clases que se emplearan para traducir la norma.

Action: contiene constructores, y métodos auxiliares para definir Action, DialogicalAction, NonDialogicalActionMethod, NonDialogicalActionPlan, Punishment, Reward y Time.

Situation: está formado por las clases Action_Situation, ActionOplSituation, Expression_Situation, Situation, Time_Expression_Situation.

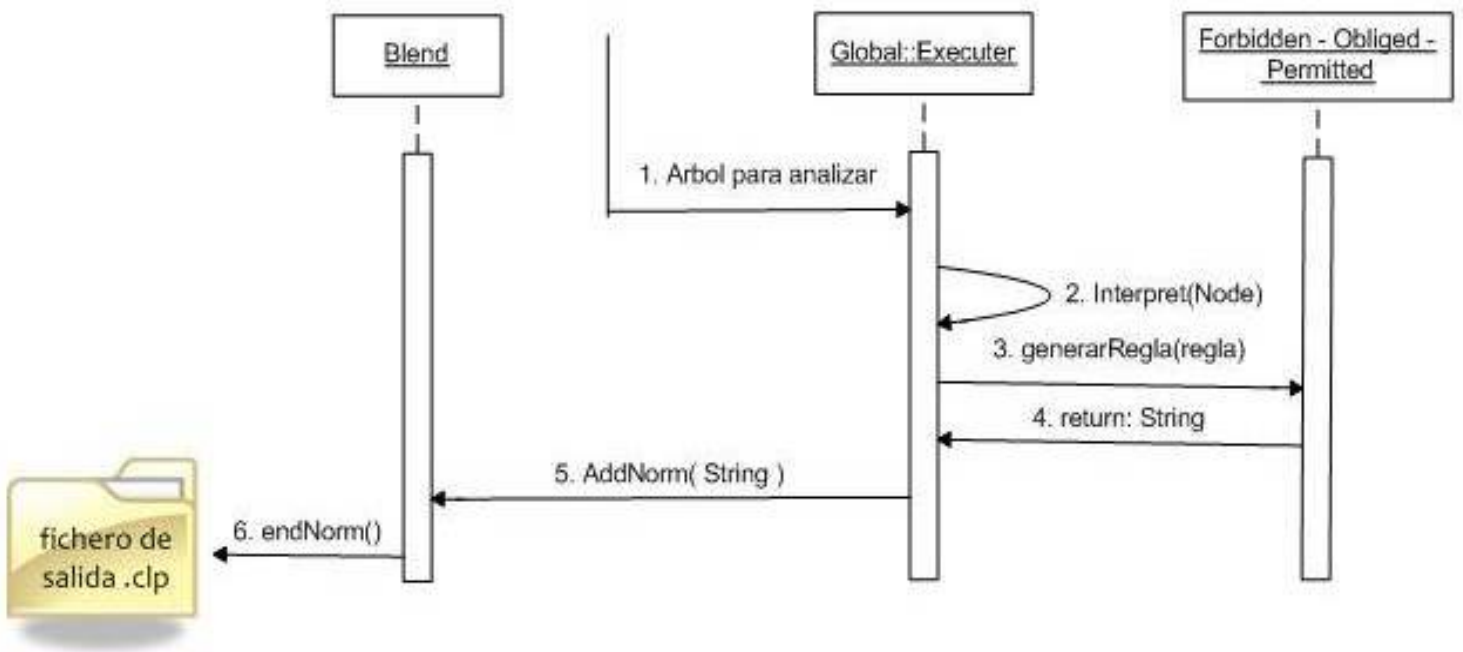
Situation.time_expresion: contiene las clases ActionTime y Date que corresponden con las expresiones de tiempo.

3.4.3. Diagramas de secuencia

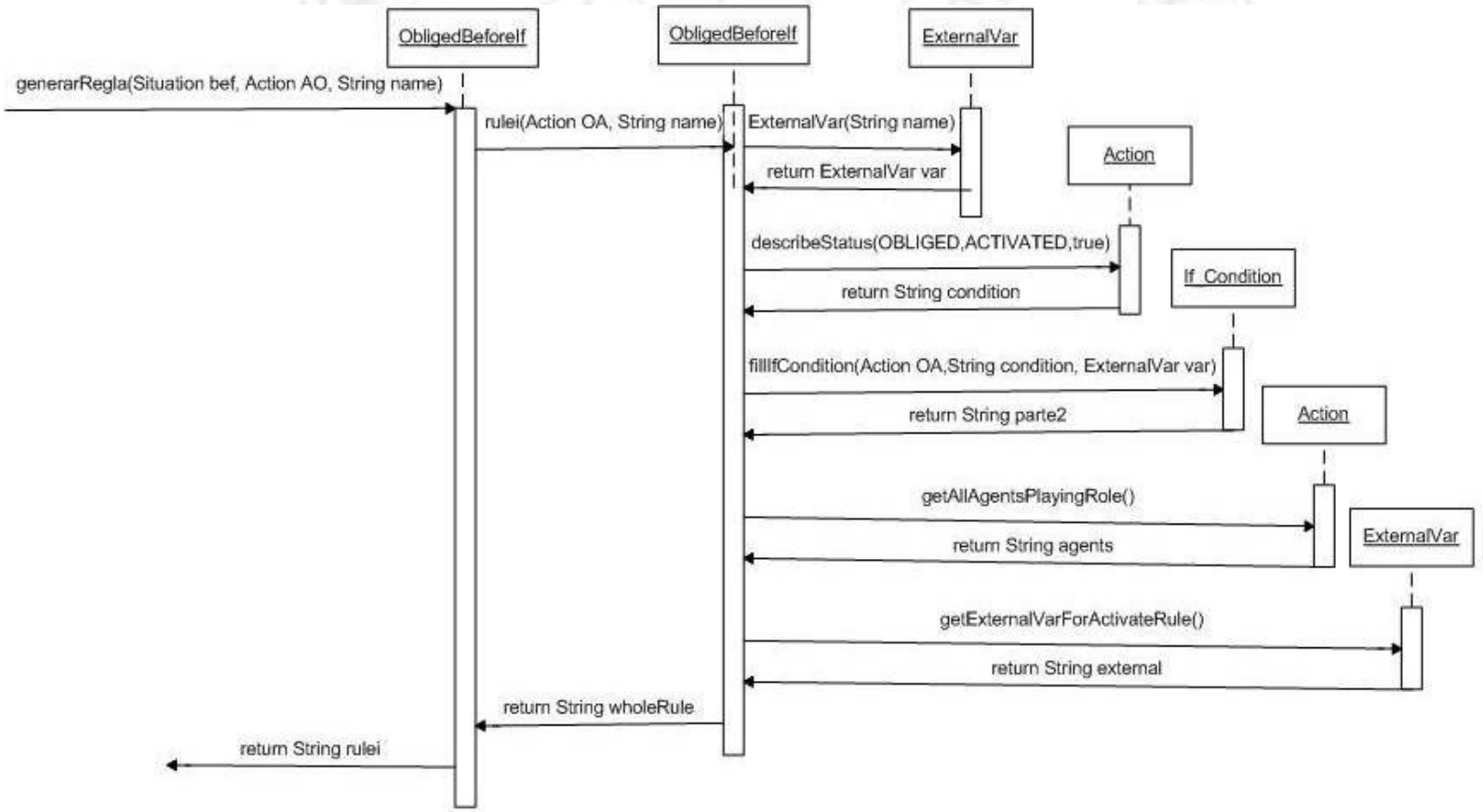
El diagrama de secuencia se utiliza para modelar la interacción entre los objetos de un sistema. Debería haber un diagrama de secuencia para cada diagrama de casos de uso, pero puesto que tenemos un único caso de uso, diseñaremos su diagrama de secuencia y mostraremos la interacción del sistema en el método de *generarRegla* para el caso concreto de *ObligedBeforeIf* que necesita las cuatro reglas.

Cabe destacar que los diagramas de secuencia se refieren al diseño de la aplicación, por lo que se describen en el lenguaje Java en el que fue creada la aplicación.

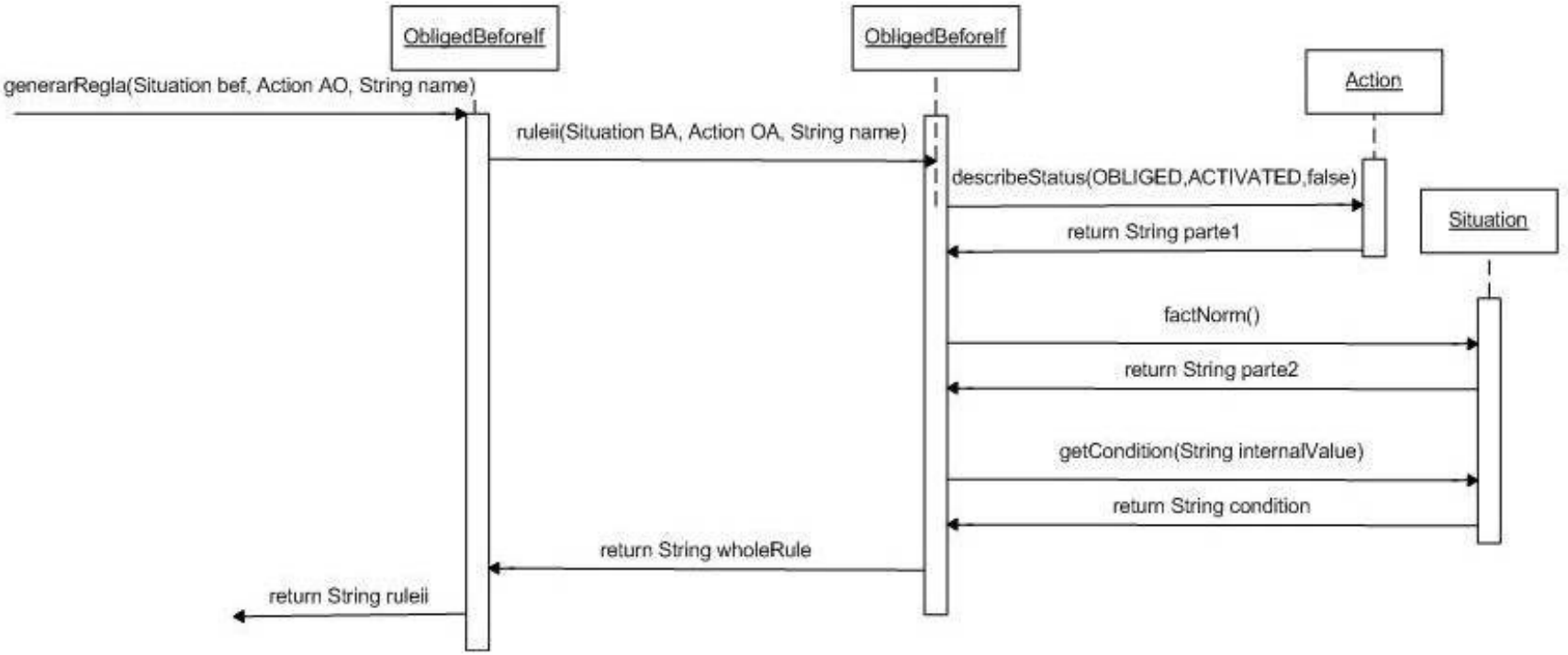
DIAGRAMA DE CASO DE USO
1. TRADUCIR NORMA



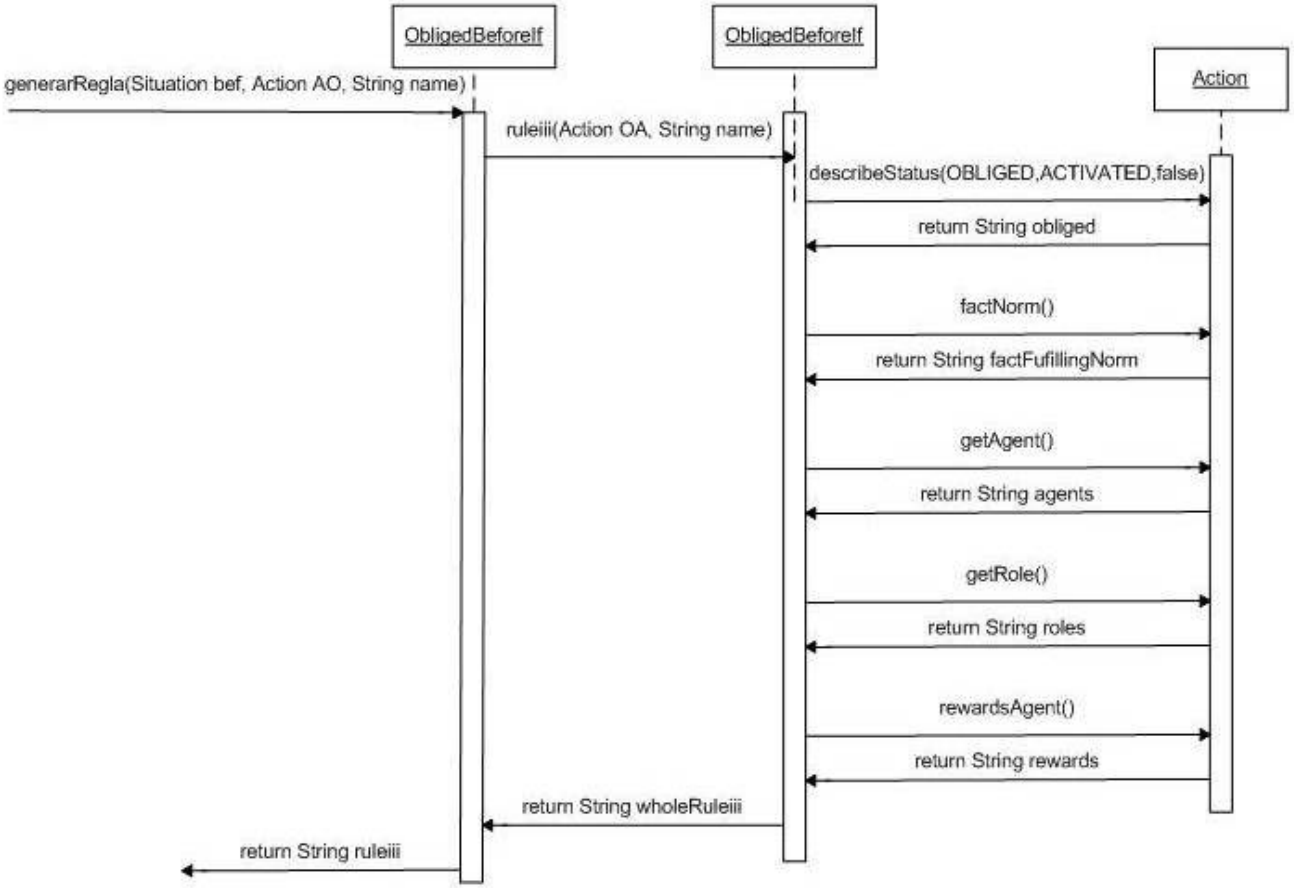
RULEI → activación de la norma



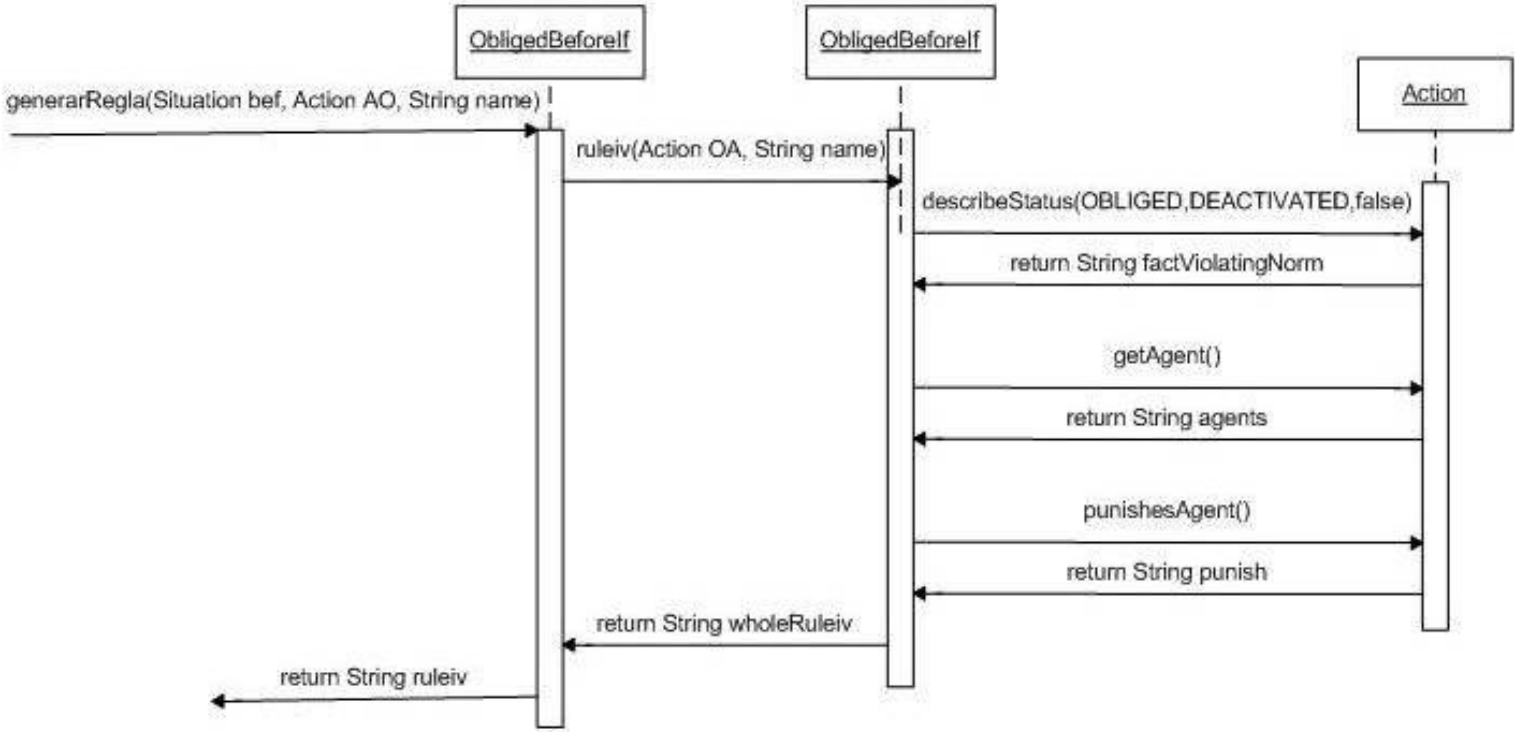
RULEII → desactivación de la norma



RULEIII → cumplimiento y recompensas



RULEIV → violación y castigos



4. Caso de estudio: sistema de donaciones y transfusiones de un hospital

4.1. Introducción

El caso de estudio a tratar consiste en un sistema de transfusiones básico. Este sistema se puede dividir en dos partes claramente diferenciadas; la correspondiente a la obtención de sangre del donante y la correspondiente a la realización de una transfusión de sangre obtenida anteriormente a un paciente.

Para la realización de una donación, el donante que ingresa en el hospital informa de que va a realizar una donación. El hospital informa al coordinador de que se quiere realizar una donación y éste asigna a un enfermero que se encuentre disponible para que realice dicha donación. Una vez realizada esta acción, el donante desaparece del sistema y una transfusión queda disponible en el sistema con el tipo de sangre del donante.

Para la realización de una transfusión, la organización, al aparecer en el entorno la sangre de un donante, al igual que con la donación, informa al coordinador de que hay una transfusión para realizar y éste vuelve a buscar un enfermero libre para que realice la transfusión. El enfermero encargado de realizar la transfusión, pregunta a cada paciente del sistema si está interesado en recibir una transfusión del tipo de sangre de la transfusión disponible. En caso afirmativo, el paciente le responde aceptando la petición con la prioridad que en ese momento tiene el paciente. En otro caso, el paciente informa al enfermero de que rechaza la transfusión. Cuando todos los pacientes responden, el enfermero le realiza la transfusión al paciente con más prioridad. Una vez terminada esta acción, tanto el paciente al que le han realizado la transfusión, como la propia transfusión desaparecen del sistema.

4.2. El sistema multi-agentes

Utilizamos el lenguaje de modelado MAS-ML para el modelado de nuestro sistema.

4.2.1. Introducción a MAS-ML

MAS-ML es un lenguaje de modelado para sistemas multi-agente. Es una extensión del meta-modelo de UML para hacer el modelado de las entidades de un sistema multi-agentes. MAS-ML extiende los diagramas de clases y de secuencia de UML y propone dos nuevos diagramas; el de organización y el de roles.

En MAS-ML se añaden nuevas meta-clases para definir relaciones de habitar, controlar, propiedad, desempeñar y para poder hacer el modelado de entidades como agentes, organizaciones, roles y entornos.

Cada elemento tendrá un modelo de representación que estará dividido en tres compartimentos. El primero será el nombre del elemento, en el central se describe las propiedades estructurales (creencias, objetivos) y en el último las propiedades dinámicas, eso es, las acciones y planes del elemento que servirán para alcanzar sus objetivos.

4.2.2. Diagramas MAS-ML de organización, roles y secuencia

Antes de mostrar los diagramas, especificaremos cada uno de los componentes del sistema utilizando la nomenclatura definida por MAS-ML.

Organización: En nuestro ejemplo sólo existe una organización, la organización principal. Esta única organización es el hospital. La organización se encarga de la creación del coordinador de enfermeros del sistema, además de actualizar el entorno informando de la llegada de un nuevo donante y de la llegada de una nueva transfusión al sistema.

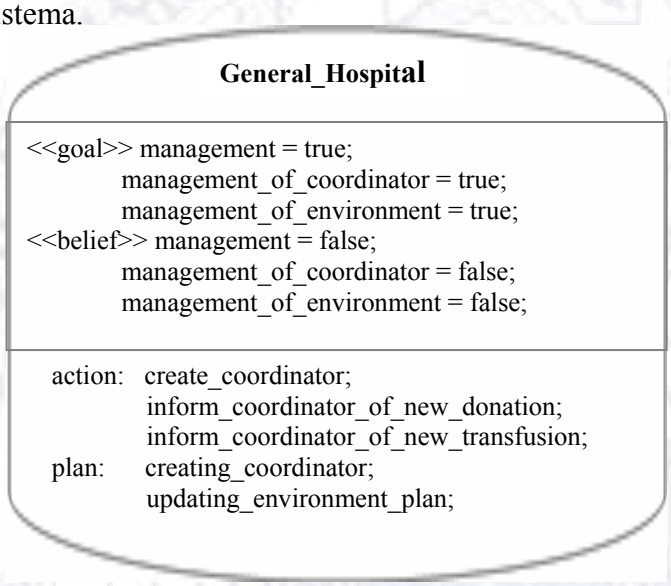


Figura 4.2.2.1 Diagrama MAS-ML de la organización

En la figura 4.2.2.1, el elemento muestra que el objetivo de la organización es el de gestión, puesto que la gestión del coordinador y del entorno son subobjetivos. Para la gestión de coordinador, basta con ejecutar el plan creating_coordinator, que va a crear al coordinador de los enfermeros. Para la gestión del entorno, la organización espera a que un donante llegue al sistema o a que encuentre una transfusión disponible en el entorno para informarle al coordinador de ese cambio en el sistema.

Roles de agentes: hemos creado un role por cada tipo de comportamiento que pueden tener los agentes del sistema.

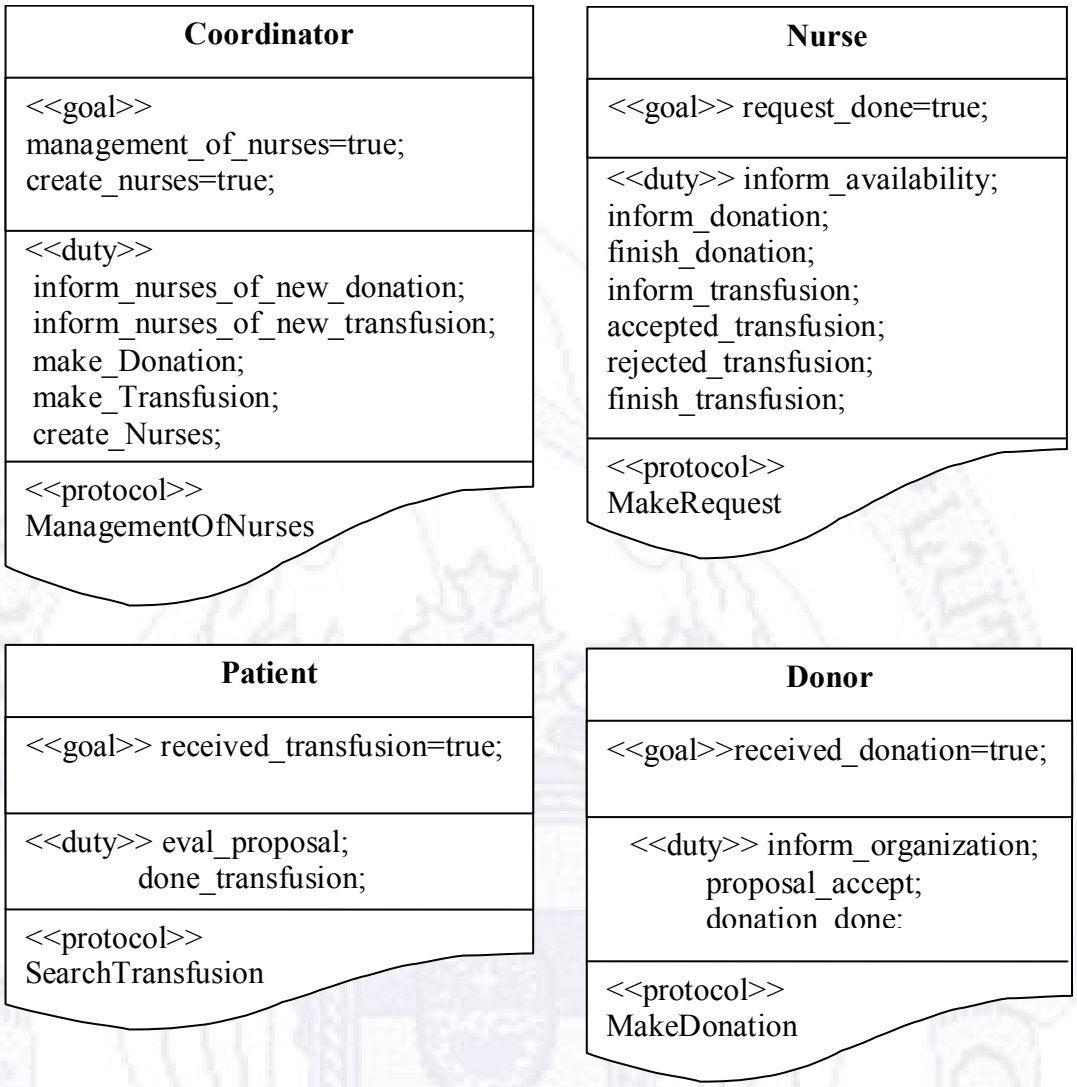


Figura 4.2.2.2 Diagrama MAS-ML de los roles del sistema

En la figura 4.2.2.2 se pueden ver los roles de agentes del sistema. Cada agente sólo va a intentar alcanzar los objetivos de los roles que desempeña. Hemos incluido todas las acciones como deberes del agente, puesto que un agente que ejecute cada uno de los roles, siempre va a implementar todas las acciones. Los protocolos de comunicación que hemos añadido indican a los agentes los mensajes que pueden recibir y la respuesta que pueden ofrecer a ese mensaje. Esto ayuda a los agentes a saber los mensajes que pueden enviar cuando les llega otro.

Role Coordinador: será encargado de la creación de los enfermeros y del mantenimiento del sistema de enfermeros mediante la asignación de la realización de una donación o de una transfusión a uno de ellos.

Role Enfermero: será el encargado de realizar las donaciones y transfusiones. También se encarga de buscar el paciente más adecuado para una transfusión.

Role Paciente: será el encargado de recibir las transfusiones por parte de un enfermero.

Role Donante: realiza la donación que recibe el enfermero que más tarde será transmitida al paciente pertinente.

Agente Doctor: Corresponde con los agentes que trabajan en el hospital. Éste tipo de agente deberá desempeñar los roles de *coordinator* o *nurse*.

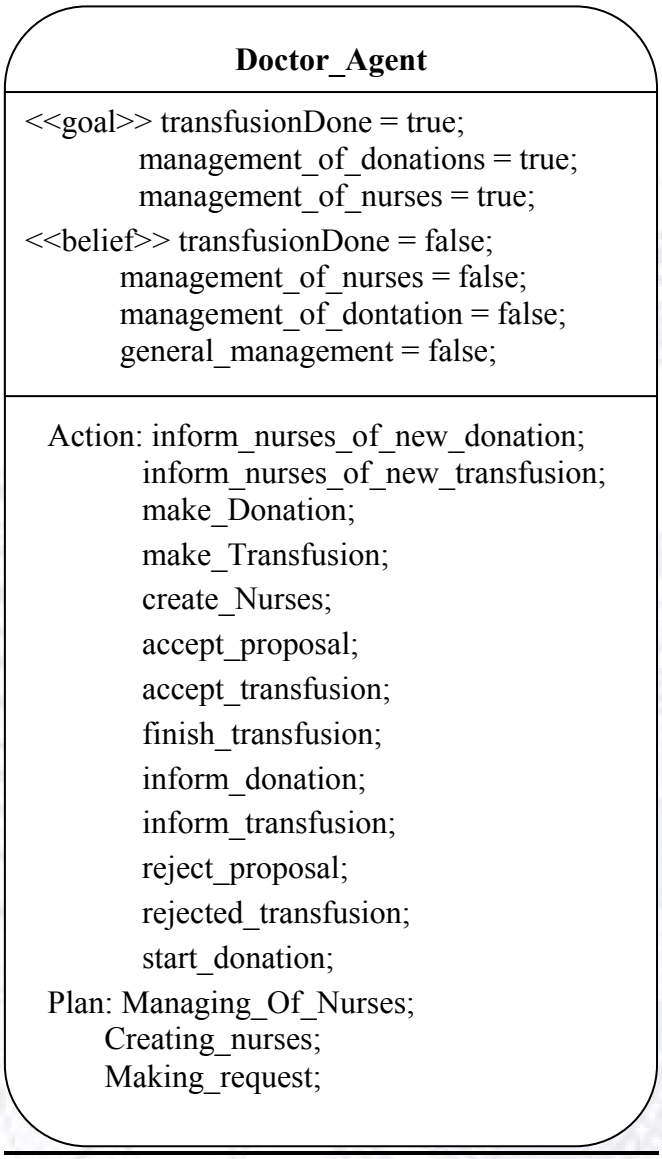


Figura 4.2.2.3 Diagrama MAS-ML del agente doctor

En la figura 4.2.2.3, el elemento muestra los objetivos que un agente del tipo doctor debe alcanzar, es decir, debe conseguir que se realicen las transfusiones, y debe dirigir las donaciones y los enfermeros. Dependiendo del plan que realice deberá ejecutar unas acciones u otras. Podríamos decir que los dos primeros planes están destinados a doctores que desempañan el role de coordinador al que le corresponderían las cinco primeras acciones y el resto de ellas junto con el último plan serían para el caso de que el agente desempeñara el role de enfermero.

Entrando un poco más en detalle para el caso de la acción create_Nurses va a ejecutar el plan creating_nurses; las cuatro primeras acciones (inform_nurses_of_new_donation, inform_nurses_of_new_transfusion, make_donation, make_transfusion) se pueden ejecutar en el plan managing_of_nurses, particularmente en este caso, las acciones a realizar son las de informar a una enfermera que se va a realizar una donación o una transfusión para que informen sobre si están disponibles o

no, y las de mandar a realizar una donación o transfusión al enfermero al que se le ha asignado la donación o la transfusión.

Si el role que desempeña el agente doctor es del tipo enfermero ejecutará la acción inform_donation para informar al donante de que va a empezar su donación en el caso de que la petición fuera una donación. Si fue una transfusión, ejecutará la acción inform_transfusion para informar a todos los pacientes de que se está realizando una transfusión.

Agente Persona: Este agente va a corresponder con los usuarios del hospital, es decir, a la hora de desempeñar un role podrá ser o el de paciente o el de donante.

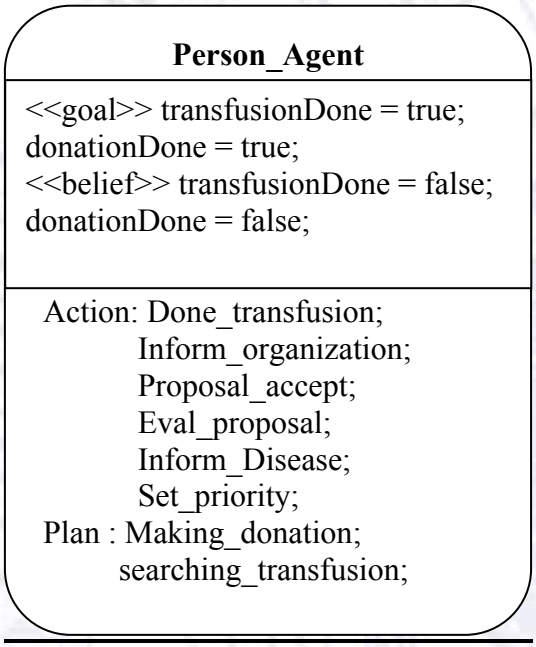


Figura 4.2.2.4 Diagrama MAS-ML del agente persona

En la figura 4.2.2.4, el diagrama muestra que los objetivos de un agente del tipo persona van a ser el de recibir una transfusión o el de realizar una donación. Para alcanzarlos, dispone de los planes searching_transfusion y making_donation.

Si desempeña el role de paciente el único plan que va a poder seguir va a ser el de searching_transfusion que va a esperar a que un enfermero indique que hay una transfusión disponible para intentar conseguirla si es compatible con la sangre del paciente.

Si por el contrario ocupa el role de donante el plan que debe seguir es making_donation, que va a informar a la organización de que quiere realizar una donación y, una vez realizada, va a salir del sistema.

Llegados a este punto en el que conocemos todos los elementos que van a formar nuestro sistema, es hora de diseñar los diagramas.

Diagrama de organización

En él se muestra cómo los roles de agentes se relacionan con la organización y cómo los agentes desempeñan sus roles dentro de la organización.

El objetivo del diagrama de organización es modelar las organizaciones del sistema y las relaciones entre ellas y otros elementos del sistema. Este diagrama modela una organización, mostrando el entorno que habita, las funciones y define los objetos, agentes y sub-organizaciones que desempeñan esas funciones. Debe haber un diagrama de organización para cada organización en el sistema.

En nuestro caso sólo tenemos una organización dentro del entorno y como consecuencia existe un único un diagrama de organización. Todos los roles están relacionados directamente a esta organización. Y cada agente podrá desempeñar alguno de los roles dentro de la organización.

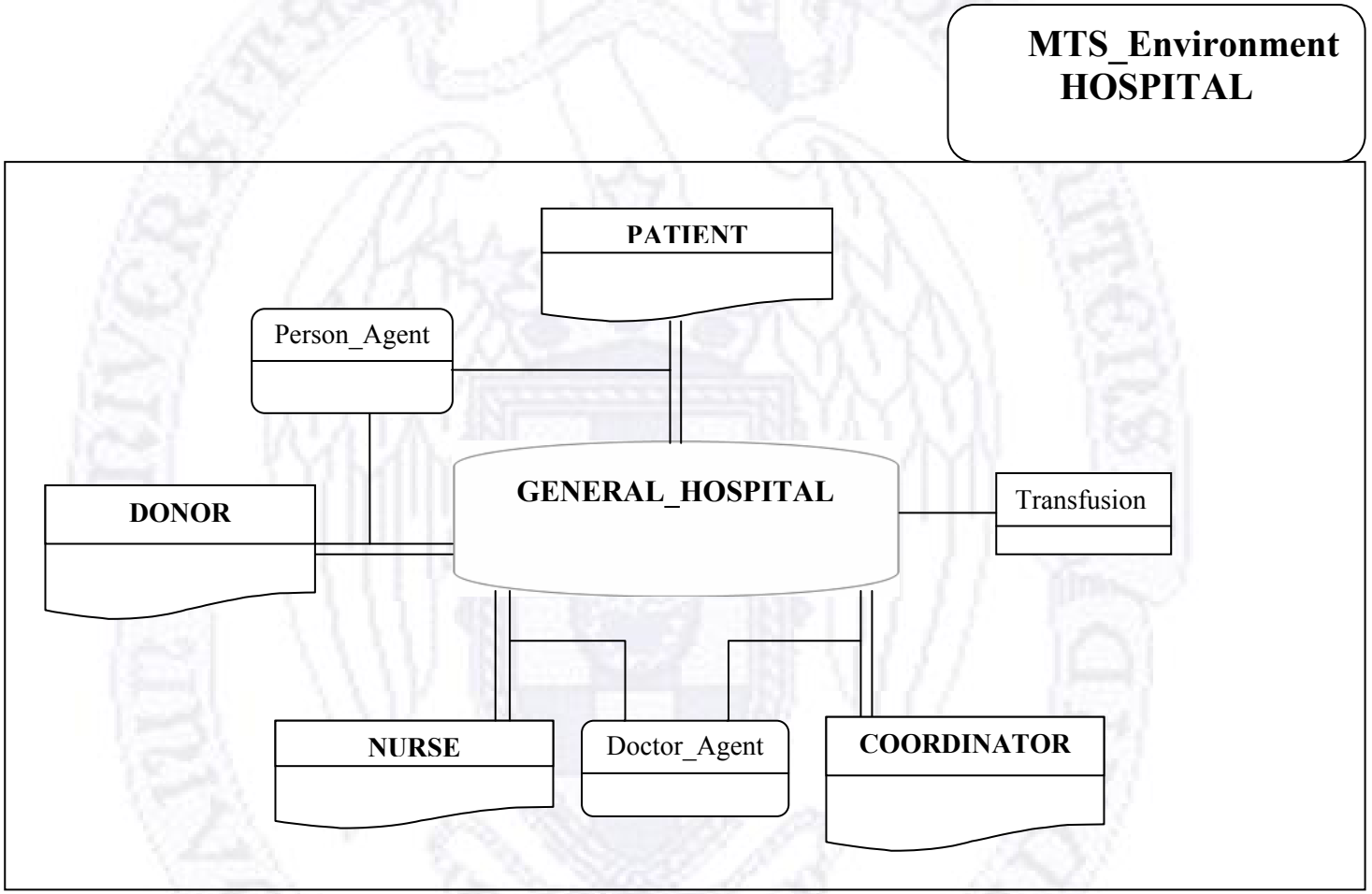


Figura 4.2.2.5 Diagrama de organización MAS-ML del sistema

Diagrama de roles de agentes:

Es necesario definirlo para ver la relación que existe entre los roles definidos en las organizaciones.

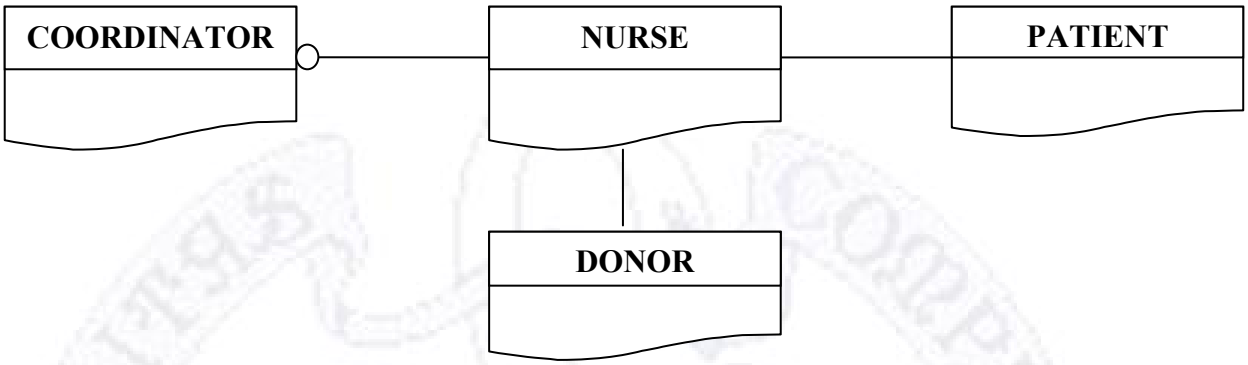


Figura 4.2.2.6 Diagrama MAS-ML de roles de agentes

Como dijimos anteriormente el coordinador debe crear los enfermeros y se encarga del mantenimiento del sistema de enfermeros mediante la asignación de la realización de una donación o de una transfusión. Por lo tanto la relación que tendrá con respecto al role de enfermeros será la de controlar.
Los enfermeros deben realizar las donaciones y transfusiones, para ello el enfermero necesita que el coordinador le haya asignado la acción correspondiente. Y el enfermero también se deberá encargar de buscar el paciente más adecuado para una transfusión.
Por lo tanto tendrá una relación de asociación con los pacientes y los donantes, el agente recibirá una donación de los donantes y realizará una transfusión a los pacientes.

Diagramas de secuencia:

Sabemos que un diagrama de secuencia en UML muestra cómo cooperan objetos dinámicos interactuando con cada uno, es decir, identifica objetos, sus interacciones y sus ejecuciones internas. Ahora MAS-ML extiende el diagrama de secuencia para entidades multi-agente como entornos, organizaciones, sub-organizaciones y agentes.

MAS-ML modela la interacción entre entidades no a través de llamadas a métodos sino con envío de mensajes. El diagrama enfoca el momento en que se envían los mensajes intercambiados entre entidades y cómo influencian los mensajes en la ejecución de las entidades. Es necesario ver todos los mensajes para entender el escenario completo.

Los dos diagramas siguientes están escritos en lenguaje de modelado MAS-ML. Arriba aparecen los agentes y las organizaciones (que tienen el formato según el tipo), con sus planes que corresponden con los rectángulos blancos de su tiempo de vida, las acciones que realizar dentro de los planes que corresponden con los rectángulos oscuros y las flechas de un agente a otro corresponden con el envío de mensajes.

DIAGRAMA DE SECUENCIA DE LA REALIZACIÓN DE UNA DONACIÓN:

Cuando un donante ingresa en el sistema, informa a la organización de que está listo para donar.

La organización le indica al coordinador que ha llegado un donante y éste pregunta a los enfermeros si están disponibles.

El coordinador espera a que le hayan respondido todos los enfermeros. Cuando le han respondido todos, escoge un enfermero que esté disponible (Si no hay ninguno, espera a que alguno lo esté).

Una vez elegido el enfermero, el coordinador le responde sólo a él de que es el encargado de hacer la donación. Esto hace que el enfermero se ponga en contacto con el donante para que sepa cuál es su enfermero y que le responda el tipo de sangre que tiene. En cuanto le responde, se realiza la transfusión y ésta está disponible en el sistema. Para terminar, el enfermero informa a la organización de que ha realizado una donación.

La organización, al recibir el mensaje, empieza a enviar los mensajes necesarios para realizar esa transfusión, pero eso se muestra en el diagrama siguiente.

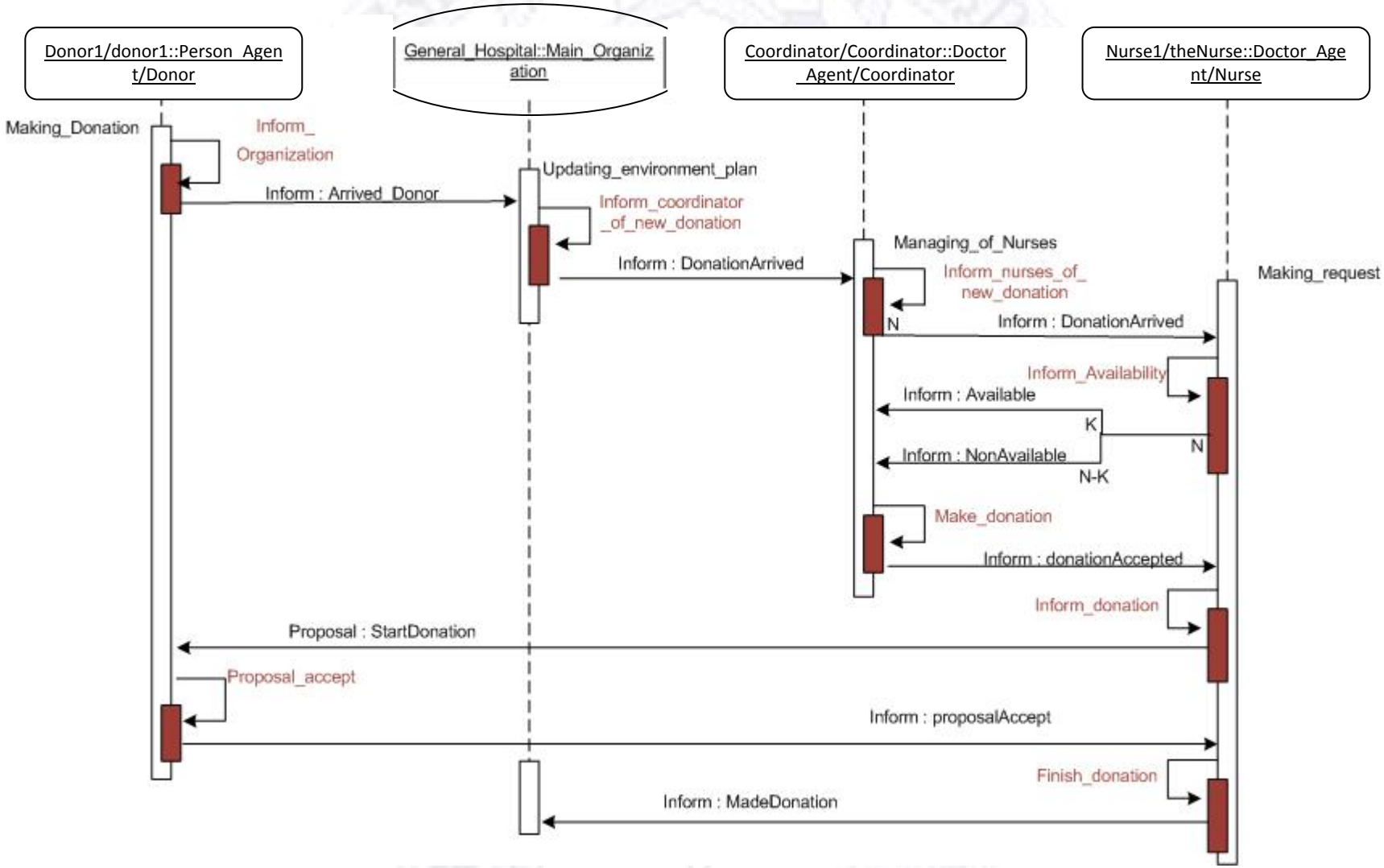

Donor1/donor1::Person_Agent/Donor
General_Hospital::Main_Organization
Coordinator/Coordinator::Doctor_Agent/Coordinator
Nurse1/theNurse::Doctor_Agent/Nurse
Making_Donation
Inform_Organization
Updating_environment_plan
Inform : Arrived_Donor
Inform_coordinator_of_new_donation
Managing_of_Nurses
Inform : DonationArrived
Inform_nurses_of_new_donation
Making_request
N
Inform : DonationArrived
Inform_Availability
Inform : Available
K
N
Inform : NonAvailable
N-K
Make_donation
Inform : donationAccepted
Inform_donation
Proposal : StartDonation
Proposal_accept
Inform : proposalAccept
Finish_donation
Inform : MadeDonation

DIAGRAMA DE SECUENCIA DE LA REALIZACIÓN DE UNA TRANSFUSIÓN A UN PACIENTE:

Este diagrama es parecido al anterior. Cuando le llega un mensaje a la organización de que hay una transfusión disponible, la organización se lo comunica al coordinador y éste pregunta a los enfermeros si están disponibles.

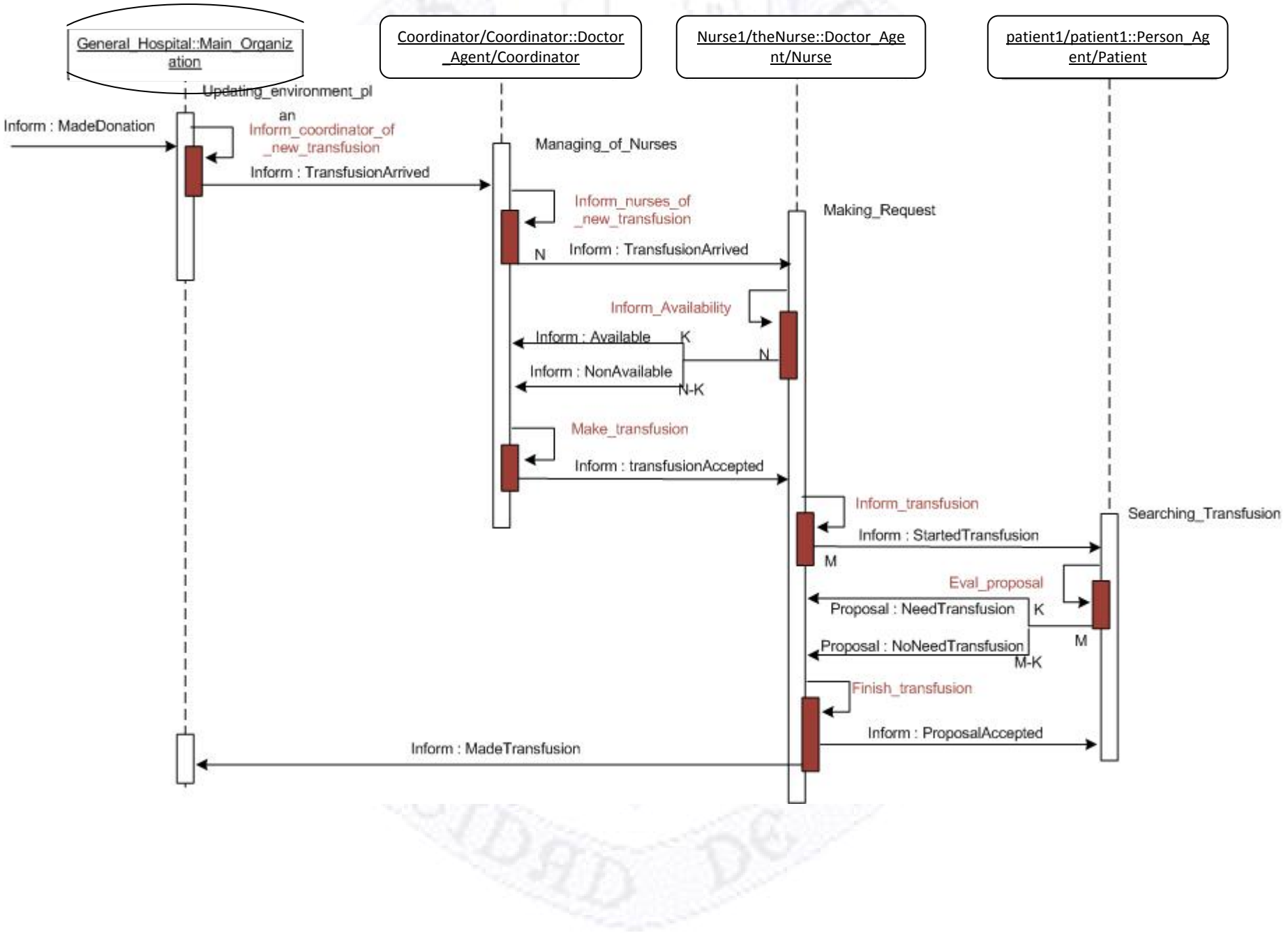
El coordinador espera a que le hayan respondido todos los enfermeros. Cuando le han respondido todos, escoge un enfermero que esté disponible (Si no hay ninguno, espera a que alguno lo esté).

Una vez elegido el enfermero, el coordinador le responde sólo a él de que es el encargado de hacer la transfusión. Esto hace que el enfermero indique a todos los pacientes que hay una transfusión disponible, además del tipo sanguíneo de esa transfusión.

Cada paciente le responde al enfermero si necesita o no esa transfusión (Si es compatible o no), y su prioridad.

En cuanto le responden todos los pacientes, se realiza la transfusión al que la necesita y tiene mayor prioridad.

Para terminar, el paciente abandona la lista de espera y el enfermero informa a la organización de que ha realizado una transfusión.


General_Hospital::Main_Organization
Coordinator/Coordinator::Doctor_Agent/Coordinator
Nurse1/theNurse::Doctor_Agent/Nurse
patient1/patient1::Person_Agent/Patient
Updating_environment_pl
an
Inform : MadeDonation
Inform_coordinator_of_new_transfusion
Inform : TransfusionArrived
Managing_of_Nurses
Inform_nurses_of_new_transfusion
Making_Request
N
Inform : TransfusionArrived
Inform_Availability
Inform : Available
K
N
Inform : NonAvailable
N-K
Make_transfusion
Inform : transfusionAccepted
Inform_transfusion
Searching_Transfusion
Inform : StartedTransfusion
M
Eval_proposal
Proposal : NeedTransfusion
K
M
Proposal : NoNeedTransfusion
M-K
Finish_transfusion
Inform : ProposalAccepted
Inform : MadeTransfusion

4.3. Las normas del sistema

4.3.1. Descripción de las normas utilizando el lenguaje

En nuestro hospital hemos incluido dos normas que los agentes conocen y deben cumplir si no quieren ser penalizados. Las normas creadas son las siguientes:

i. La primera de ellas es que **los enfermeros no pueden realizar una acción, ya sea una transfusión o una donación, si no ha pasado más de un minuto desde la última que realizaron.** Si el agente cumple la norma, se le informa de que puede volver a realizar otra acción. En el caso de que no la cumpla, el agente es despedido del hospital. La norma descrita en la gramática ya explicada anteriormente en este documento es la siguiente:

```
Wait1Minute
FORBIDDEN {(
UTTER( transfusion; available; inform(S:; SR:Nurse; R:_;
RR:organization[RT_; CO:_; LA:_;  EN:_; ON:_; PR:_; ID:_;
RW:_; IR:_; RB:_]) )
  BETWEEN (
UTTER( Transfusion; MadeTransfusion; inform (S:; SR:Nurse;
R:_;  RR:organization  [RT:_;  CO:_;  LA:_;  EN:_;  ON:_;
PR:MakeTransfusion; ID:update_environment?nurse; RW:_; IR:_;
RB:_]) ),
 1 MINUTES OF UTTER( Transfusion; MadeTransfusion;
MadeTransfusion (S:; SR:Nurse; R:_; RR:organization [ RT:_;
CO:_;     LA:_;     EN:_;     ON:_;     PR:MakeTransfusion;
ID:update_environment?nurse; RW:_; IR:_; RB:_]) )
  )
)}
```

ii. La segunda de ellas consiste en la **prohibición de que los pacientes mientan con respecto a la gravedad de su enfermedad**. Si el nivel de prioridad que dicen no es cierto, se les aplicará un castigo, que es eliminarle de la lista de pacientes que esperan una transfusión de sangre. Esta norma se puede llevar a cabo porque la organización introduce en el sistema Jess la prioridad del paciente cuando este ingresa en el sistema. La norma descrita en la gramática ya explicada anteriormente en este documento es la siguiente:

```
RemoveLiars
FORBIDDEN {(
  UTTER(  donation;  askingPriority;  INFORM  (S:AGENT;
  SR:patient; R:_; RR:nurse [ RT:_; CO:_; LA:_; EN:_; ON:_;
  PR:_; ID:_; RW:_; IR:_; RB:_]) )
  IF agentClass :: AGENT . priority <> CONTENT
)}
```

4.3.2. Reglas en Jess para cada norma

A continuación definimos las normas descritas en el apartado anterior utilizando el lenguaje de programación Jess.

1. **Norma para controlar la acción de hacer transfusión de los enfermeros: Esta norma está definida en Jess por 4 reglas:**

i.

```
;(rule i)
(defrule Wait1Minute_rule_i
      ?currentTime <- (currentTime)
      ?factActivatingNorm <- (dialogical-action
            (scene Transfusion)(stateMadeTransfusion)
            (performative inform)(sRole Nurse)
            (rRole organization)
            (protocol MakeTransfusion)
            (conversation-id update_environment?nurse))
      =>
      (assert (FORBIDDEN-dialogical-action
            (scene transfusion)
            (state available)(performative inform)
            (sender (fact-slot-value
                  ?factActivatingNorm sender))
            (sRole Nurse)(rRole organization)
            (status ACTIVATED))
))
```

La regla i indica que la norma se activa cuando una enfermera ejecuta la acción de realizar una transfusión

ii.

```
;(rule ii)
(defrule Wait1Minute_rule_ii
      ?currentTime <- (currentTime)
      ?forbidden <-(FORBIDDEN-dialogical-action
            (scene transfusion)(state available)
            (performative inform)(sRole Nurse)
            (rRole organization)(status ACTIVATED))
      ?factActivatingNorm <- (dialogical-action
            (scene Transfusion)
            (state MadeTransfusion)
            (performative inform)
            (sRole Nurse)
            (rRole organization)
            (protocol MakeTransfusion)
            (conversation-id update_environment?nurse))
      =>
      (bind ?date (time))
      (bind ?timeAction (+ (fact-slot-value
      ?factActivatingNorm time) 20))
            (if (>= ?date ?timeAction) then
                  (modify ?forbidden (status DEACTIVATED))
))
```

La regla ii indica que la norma se desactivará cuando pase un minuto de la acción realizada por la enfermera, que es la misma que activó la norma.

iii.

```
;(rule iii)
```

```
(defrule Wait1Minute_rule_iii
      ?currentTime <- (currentTime)
      ?factFulfillingNorm <-(FORBIDDEN-dialogical-action
            (scene transfusion)(state available)
            (performative inform)(sRole Nurse)
            (rRole organization)(status DEACTIVATED))
      =>
      (bind ?result (assert (NormStatus_per_Agent (agent
      (fact-slot-value ?factFulfillingNorm sender)) (norm
      ?factFulfillingNorm)
      (status VIOLATED))))
      (try
            (= false ?result)
      catch
            (modify ?result (status FULFILLED))
            (assert (REWARD (norm ?factFulfillingNorm)
                  (promoter organization)
                  (rewardedAgent (fact-slot-value
                        ?factFulfillingNorm sender))
                  (reward "Nurse Free")))
))
```

La regla iii indica que si la norma está desactivada y no la ha violado, se cumple la norma y se añade a la base de hechos la recompensa para la enfermera, que en este caso es que queda libre para realizar otra acción.

iv.

```
;(rule iv)
(defrule Wait1Minute_rule_iv
      ?currentTime <- (currentTime)
      ?forbidden <-(FORBIDDEN-dialogical-action
                  (scene transfusion)(state available)
                  (performative inform)(sRole Nurse)
                  (rRole organization)(status ACTIVATED))
      ?factViolatingNorm <- (dialogical-action
                  (scene transfusion)(state available)
                  (performative inform)(sRole Nurse)
                  (rRole organization))
      =>
      (if (= (fact-slot-value ?factViolatingNorm sender)
            (fact-slot-value ?forbidden sender)) then
            (assert (NormStatus_per_Agent
                  (agent (fact-slot-value
                        ?factViolatingNorm sender))
                  (norm (fact-id ?forbidden))
                  (status VIOLATED)
                   (reason (fact-id ?factViolatingNorm))))
            (assert (PUNISHMENT (norm ?factViolatingNorm)
                  (authority organization)
                  (punishedAgent (fact-slot-value
                        ?factViolatingNorm sender))
                  (punishment "Fire nurse")))
))
```

La regla iv indica que si la norma está activada y la enfermera realiza otra acción, se incumple la norma y se añade a la base de hechos la penalización para la enfermera, que en este caso es que va a ser despedida.

2. **Norma para controlar que los pacientes no mienten en su prioridad:** Esta norma únicamente tiene las reglas i y iv, la regla ii y iii no existen puesto que no se va desactivar nunca y, por ello, tampoco se puede saber si ha cumplido la norma. La regla iv comprueba que la prioridad, que es el contenido de la acción, es la misma que la prioridad que dio la organización en el momento en que el paciente entró en el sistema. Si no ha sido así, activa la regla iv y se inserta la penalización del agente en la base de hechos.

i.

```
;(rule i)
(defrule patientNorm_rule_i
    ?currentTime <- (currentTime)
    =>
    (assert (FORBIDDEN-dialogical-action
          (scene donation)(state askingPriority)
          (performative INFORM)(sRole patient)
          (rRole nurse)(status ACTIVATED))
    ))
```

La regla i simplemente activa la norma, ya que va a estar activa desde el comienzo del sistema.

ii. La regla ii no existe puesto que no se va desactivar nunca

iii. La regla iii tampoco existe puesto que no se puede saber si ha cumplido la norma.

iv.

```
;(rule iv)
(defrule patientNorm_rule_iv
      ?currentTime <- (currentTime)
      ?forbidden <-(FORBIDDEN-dialogical-action
            (scene donation)(state askingPriority)
            (performative INFORM)(sRole patient)
            (rRole nurse)(status ACTIVATED))
      ?factViolatingNorm <- (dialogical-action
            (scene donation)(state askingPriority)
            (performative INFORM)(sRole patient)(rRole
            nurse))
      ?attrib <- (attribute-value (class agentClass)
            (attribute priority ))
=>
      (bind ?agentAttrib (fact-slot-value ?attrib
            objectORagent))
      (bind ?valueAttrib (fact-slot-value ?attrib value))
      (bind ?agentSender (fact-slot-value
            ?factViolatingNorm sender))
      (if (= ?agentAttrib ?agentSender) then
            (bind ?contents (fact-slot-value
            ?factViolatingNorm content))
            (foreach ?content (list ?contents)
            (if (<> ?content ?valueAttrib) then
                  (assert (NormStatus_per_Agent
                  (agent (fact-slot-value
                  ?factViolatingNorm sender))
                  (norm (fact-id ?forbidden))
                  (status VIOLATED)
                  (reason (fact-id ?factViolatingNorm))
                  ))
            ))
```

)
)

La regla iv comprueba que la prioridad, que es el contenido de la acción, es la misma que la prioridad que dio la organización en el momento en que el paciente entró en el sistema. Si no ha sido así, se activa la regla y se inserta la penalización del agente en la base de hechos.

4.4. Implementación utilizando ASF

Utilizamos el framework ASF para hacer la implementación de este sistema.

4.4.1. Introducción a ASF

ASF es un framework orientado a objetos para implementar sociedades de agentes. Las sociedades de agentes son sistemas multi-agentes en las cuales es requerido o importante representar agentes desempeñando roles en organizaciones. Usando este framework, es posible implementar varios agentes; cada uno ejecutando diferentes roles en diferentes organizaciones. El framework contempla agentes, roles, organizaciones y entornos como entidades de primer orden. Para cada entidad, se describe un ciclo de vida detallado mediante los estados de ejecución y los eventos que causan la (permitida) transición de la entidad desde un estado a otro. Basado en representaciones gráficas de los modelos de ciclo de vida, los modelos computacionales de cada entidad pueden ser explorados. Los modelos computacionales definen el comportamiento de las entidades asociadas con cada estado descrito en los modelos de ciclo de vida.

El framework ASF soporta la implementación de agentes, roles, organizaciones y entornos como abstracciones de primer orden. Sin embargo, las entidades de entornos, organizaciones, agentes y roles que son usadas en MAS no están disponibles en sistemas orientados a objetos. Como consecuencia directa, no es posible mapear directamente ninguna de estas entidades en una clase orientada a objetos porque estas entidades y clases tienen diferentes elementos.

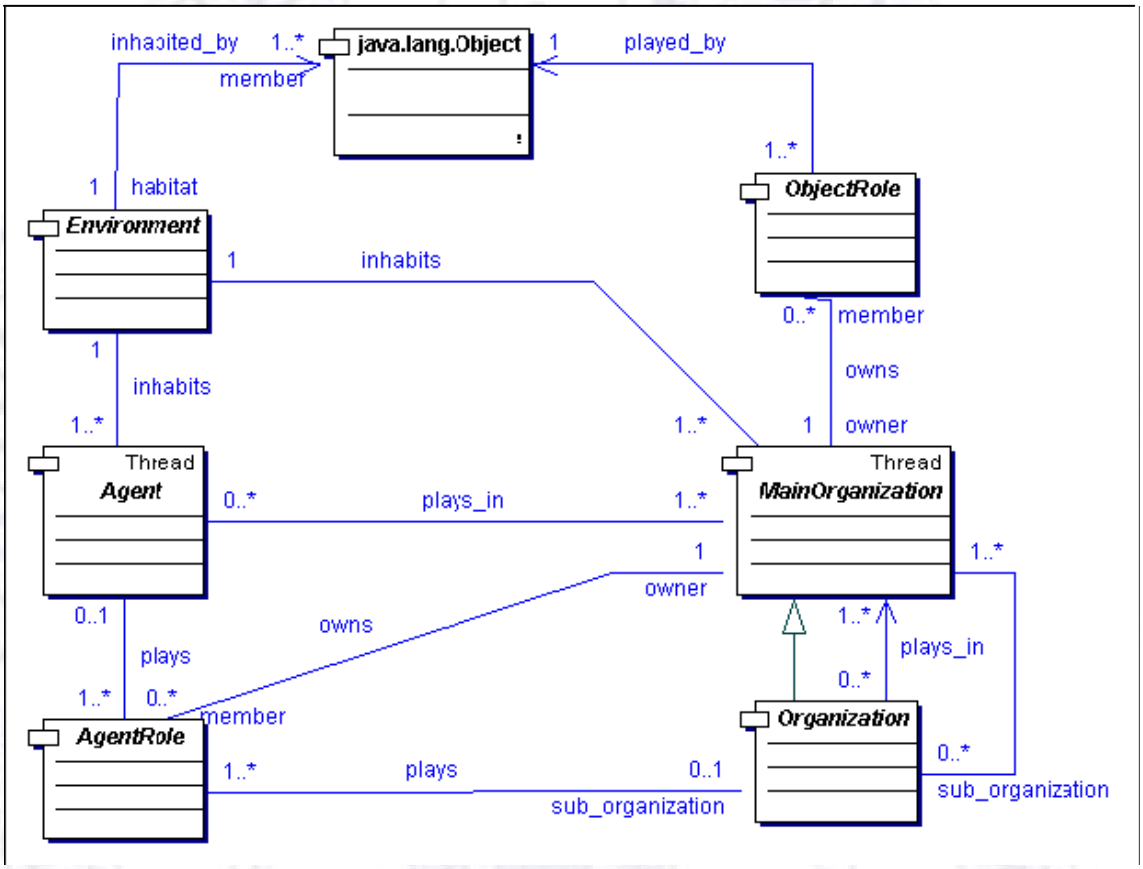
Para implementar entidades MAS usando un lenguaje de programación orientado a objetos como Java es necesario crear conjuntos de clases que representan las nuevas entidades. El framework está compuesto de una clase abstracta que representa la entidad y otras clases abstractas que representan las propiedades de la entidad.

4.4.2. Diagramas de clases del ASF

Para componer el framework es importante entender cómo los módulos están relacionados. Las relaciones entre los módulos están representadas por las relaciones entre las clases abstractas asociadas con agente, organización, rol de agente, rol de objeto y entorno.

Los entornos son los hábitats de los agentes, organizaciones y objetos. Además, la clase abstracta Environment está asociada con las clases abstractas Agent y MainOrganization y con la clase de Java Object. La clase Environment define atributos para almacenar agentes, organizaciones y objetos.

Los agentes interpretan roles en las organizaciones. La clase abstracta Agent está relacionada a la clase abstracta AgentRole para representar los roles que los agentes representan y a la clase abstracta MainOrganization para representar las organizaciones donde los agentes interpretan roles. La clase AgentRole está también asociada con la clase Organization para indicar organizaciones que pueden interpretar roles y con la clase MainOrganization para indicar poseedores de el rol. La clase MainOrganization está también relacionada a la clase ObjectRole para definir los poseedores de roles de objetos. Por último, la clase ObjectRole está relacionada a la clase Object para identificar los objetos que interpretan roles.



Una vez vistos todos los módulos del sistema, ahora queda por ver como se relacionan unos con otros.

En la figura siguiente, el entorno del sistema está relacionado con las transfusiones y con los agentes, puesto que todos los agentes y todas las transfusiones disponibles están almacenados en el entorno.

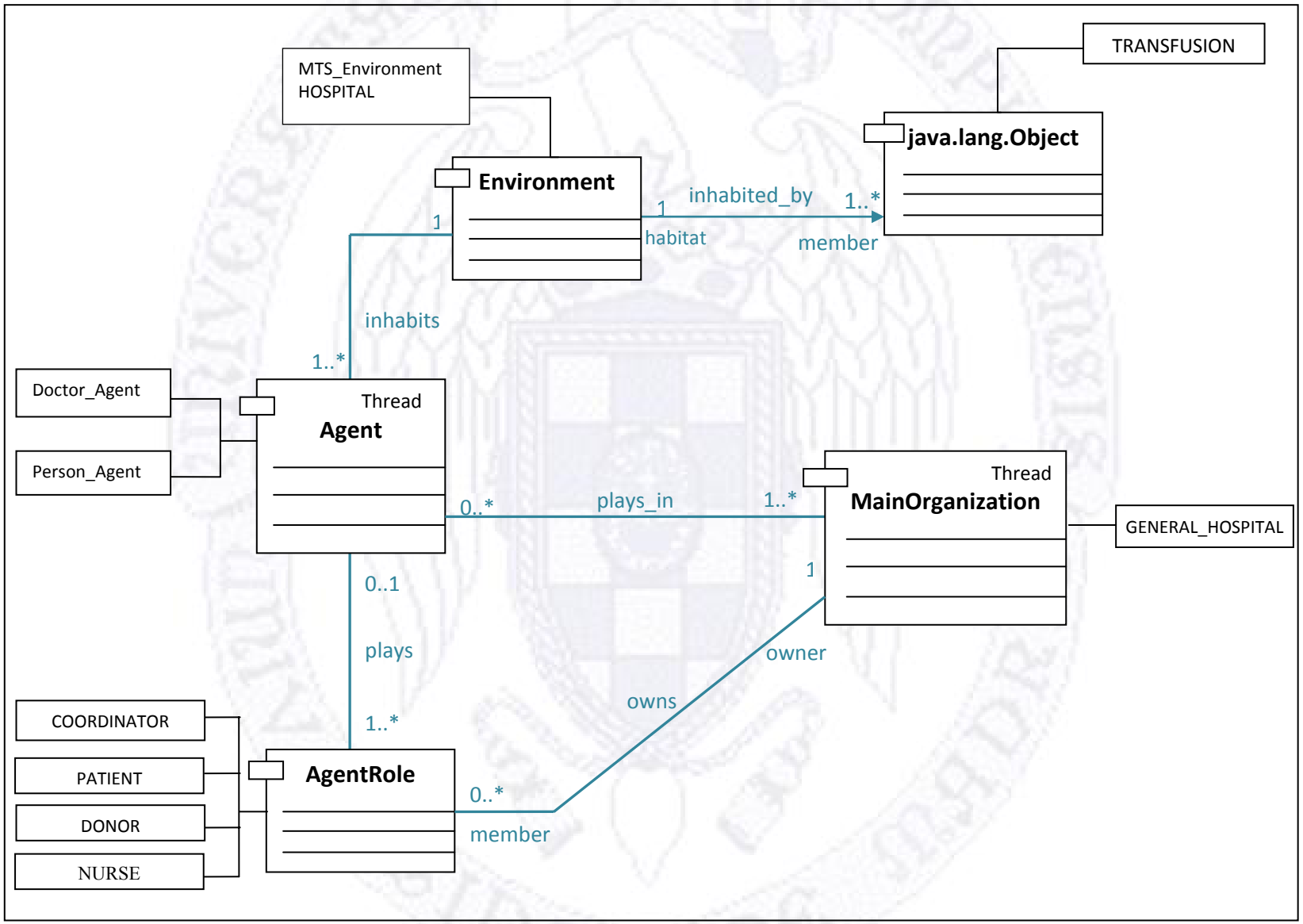
Se muestra cómo los agentes están relacionados con los roles. Están representados los roles que los agentes pueden desempeñar, y la organización principal que corresponde con la única organización que tiene el sistema donde los agentes desempeñan los roles.

El entorno está relacionado con los objetos, en el entorno existe un objeto Transfusion.

Como mencionamos anteriormente el sistema tiene una única organización, la clase General_Hospital que corresponde con la clase MainOrganization.

La clase Doctor_Agent y Person_Agent aparecen en el módulo agente extendiendo la clase Agent.

La clase AgentRole será extendida por cuatro clases que desempeñan los posibles comportamientos que los agentes podrán tener en el sistema, Coordinator, Nurse, Donor y Patient.


MTS_Environment
HOSPITAL
TRANSFUSION
java.lang.Object
Environment
1
inhabited_by
1..*
habitat
member
1
inhabits
1..*
Doctor_Agent
Person_Agent
Thread
Agent
0..*
plays_in
1..*
Thread
MainOrganization
GENERAL_HOSPITAL
0..1
plays
1..*
1
owner
owns
COORDINATOR
PATIENT
DONOR
NURSE
AgentRole
0..*
member

4.4.3. Código para explicar la implementación

Para entender del todo el diseño del sistema, ahora incluimos las implementaciones escritas en lenguaje Java de algunas de las especificaciones vistas en el punto anterior.

```
public class General_Hospital extends MainOrganization{
  public General_Hospital {
    //Añadir creencias del agente
    this.beliefs.add (new LeafBelief ("boolean", "management_of_environment", false"));
    this.beliefs.add (new LeafBelief ("boolean", "management_of_coordinator", "false"));
    this.beliefs.add (new LeafBelief ("boolean", "management", "false"));
    //Añadir objetivo principal del agente
    objectGoal = new CompositeGoal ("boolean", " management", "true");
    objectGoal.setPriority (1);
    //Añadir primer subobjetivo del agente y su plan para alcanzarlo
    objectSubGoal = new LeafGoal ("boolean", " management_of_coordinator", "true");
    objectSubGoal.setPriority (2);
    objectPlan = new creating_coordinator();
    objectSubGoal.setPlan (objectPlan);
    objectPlan.setGoal (objectSubGoal);
    this.plans.add (objectPlan);
    objectGoal.setSubGoal(objectSubGoal);
    //Añadir segundo subobjetivo del agente y su plan para alcanzarlo
    objectSubGoal = new LeafGoal ("boolean", " management_of_coordinator", "true");
    objectSubGoal.setPriority (2);
    objectPlan = new creating_coordinator();
    objectSubGoal.setPlan (objectPlan);
    objectPlan.setGoal (objectSubGoal);
    this.plans.add (objectPlan);
    objectGoal.setSubGoal(objectSubGoal);
    this.objects.add(objectGoal);
}}
```

Figura 4.4.3.1 Código ASF para asignar objetivos y planes a la organización

En la figura 4.4.3.1 se muestra el código que hay que añadir para implementar la organización. Primero se añaden las creencias que va a tener al inicio y luego vamos añadiendo sus objetivos con cada plan que queremos que ejecute para alcanzarlo.

```
public class Person_Agent extends Agent implements Serializable{
public Person_Agent {
        //Añadir creencias del agente
        this.beliefs.add (new LeafBelief ("boolean","transfusionDone", "false"));
        this.beliefs.add (new LeafBelief ("boolean","donationDone", "false"));
        //Añadir primer objetivo del agente y su plan para alcanzarlo
        objectGoal = new LeafGoal ("boolean", "transfusionDone", "true");
        objectGoal.setPriority (1);
        objectPlan = new Searching_transfusion ();
        objectPlan.setGoal (objectGoal);
        objectGoal.setPlan (objectPlan);
        this.plans.add (objectPlan);
        this.goals.add(objectGoal);
        //Añadir segundo objetivo del agente y su plan para alcanzarlo
        objectGoal = new LeafGoal ("boolean", " donationDone", "true");
        objectGoal.setPriority (1);
        objectPlan = new Making_donation ();
        objectGoal.setPlan (objectPlan);
        objectPlan.setGoal (objectGoal);
        this.plans.add (objectPlan);
        this.goals.add(objectGoal);
}}
```

Figura 4.4.3.2 Código ASF para asignar objetivos y planes a las personas

En la figura 4.4.3.2 se muestra el código que hay que añadir para que un agente, en este caso del tipo persona, se comporte como nosotros queremos. Primero añadimos las creencias que va a tener al inicio y luego vamos añadiendo sus objetivos con cada plan que queremos que ejecute para alcanzarlo. En este caso, lo que queremos es que primero realice transfusiones y luego donaciones dependiendo del role que desempeñe dentro de la organización.

4.5. Ejecución del sistema

4.5.1. Descripción de la incorporación de Jess a la implementación en ASF en el ejemplo

Para incorporar Jess al ejemplo, hemos utilizado las librerías que proporciona la compañía Sandia, creadora de Jess, que permiten la asociación de un sistema Jess en una aplicación Java.

De esta forma, una vez creadas las normas con nuestra aplicación, podemos añadir hechos a la base de datos de Jess de las acciones realizadas por los agentes que nos puedan interesar porque pueden hacer que se viole o se cumpla una determinada norma de nuestro sistema. Para hacer que nuestra aplicación sea un sistema seguro, únicamente añaden cada una de estas acciones los agentes que no están implicados en esas normas y que no tienen intereses que les hagan mentir para su propio beneficio.

Para comprobar si un agente ha violado o ha cumplido una norma, el sistema Jess informa mediante eventos a la parte Java de nuestra aplicación. Una vez que el sistema Jess lanza un evento, se informa a la organización de ese cumplimiento o esa violación de la norma para que emprenda las acciones correspondientes para cada caso.

Para ver el funcionamiento del sistema Jess, comprobaremos el impacto de cada norma incluida en nuestro sistema:

- **Norma de prohibición de la realización de dos acciones en menos de un minuto por parte de un enfermero:**
  Para el correcto funcionamiento de la norma, ha sido necesario incluir en el sistema Jess las acciones correspondientes de añadir al sistema un enfermero, añadir una transfusión realizada por un enfermero determinado y la respuesta sobre la disponibilidad de un enfermero. Para que los enfermeros no nos puedan engañar según sean sus intereses, estas acciones únicamente las añaden la organización, que se encarga de la primera, o el coordinador, que se encarga de las dos siguientes.
  Si ha cumplido o ha violado la norma, se informa a la organización que, respectivamente, informa al enfermero de que puede volver a realizar una transfusión o le elimina del sistema por no haber respetado la norma.

- **Norma de prohibición que un paciente mienta sobre su prioridad:**
  Ha sido necesario el poder incluir en el sistema Jess los hechos que incluyen la prioridad dada al paciente por parte de la organización y los que incluyen la prioridad con la que el paciente ha respondido al enfermero para conseguir una transfusión. Como el paciente no puede ser el encargado de añadir estas acciones en el sistema Jess, puesto que si ha mentido sobre su prioridad probablemente vuelva a mentir, son la organización y los enfermeros los encargados de incluir en el sistema dichas acciones. La organización añade la acción sobre la prioridad que ha dado al paciente y el enfermero añade la acción sobre la prioridad con la que le ha respondido el paciente.
  Si ha cumplido la norma, no se realiza ninguna acción, pero en el que caso de que la haya violado, se informa a la organización, que es la encargada de eliminarle del sistema y volver a poner la transfusión en el sistema a la espera de encontrar a otro paciente, en el caso de que el paciente que ha violado la norma hubiera ganado esa transfusión. Si no la hubiera conseguido, el paciente que la consiguió se la queda si este agente no violó la norma.

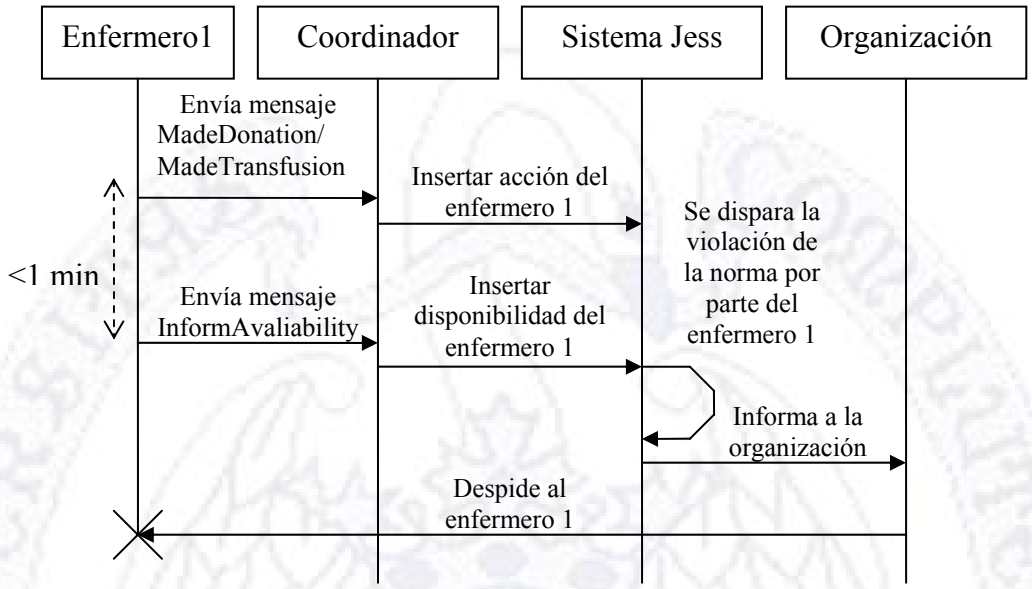
4.5.2. Escenarios de ejecución del sistema

Para comprender mejor el comportamiento de nuestro sistema con respecto a sus normas, vamos a explicar los pasos que siguen en nuestro sistema desde que el agente viola una norma hasta que el sistema efectúa las acciones necesarias para que ese agente sea penalizado.

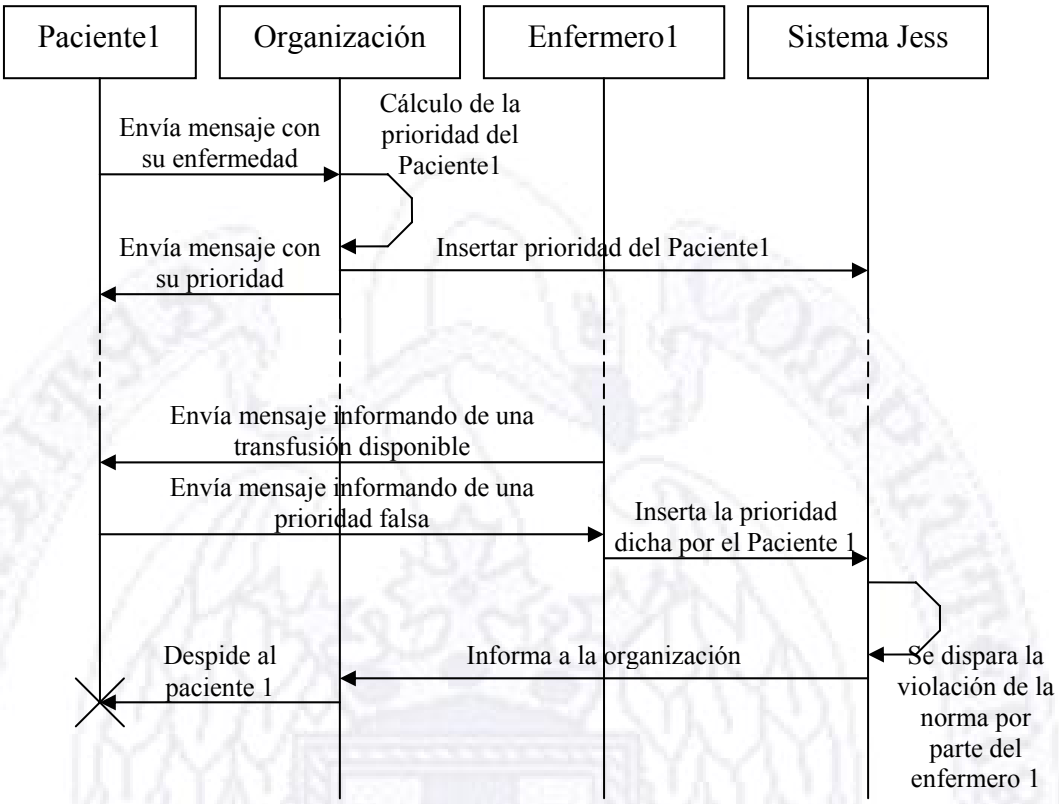
Vamos a simular los escenarios de la violación de una regla por parte de un enfermero y por parte de un paciente:

i. **Un enfermero viola una norma.** Si el coordinador informa al sistema de la disponibilidad de un enfermero y éste acaba de realizar otra acción, el sistema Jess se da cuenta de que el agente ha violado la norma e informa a

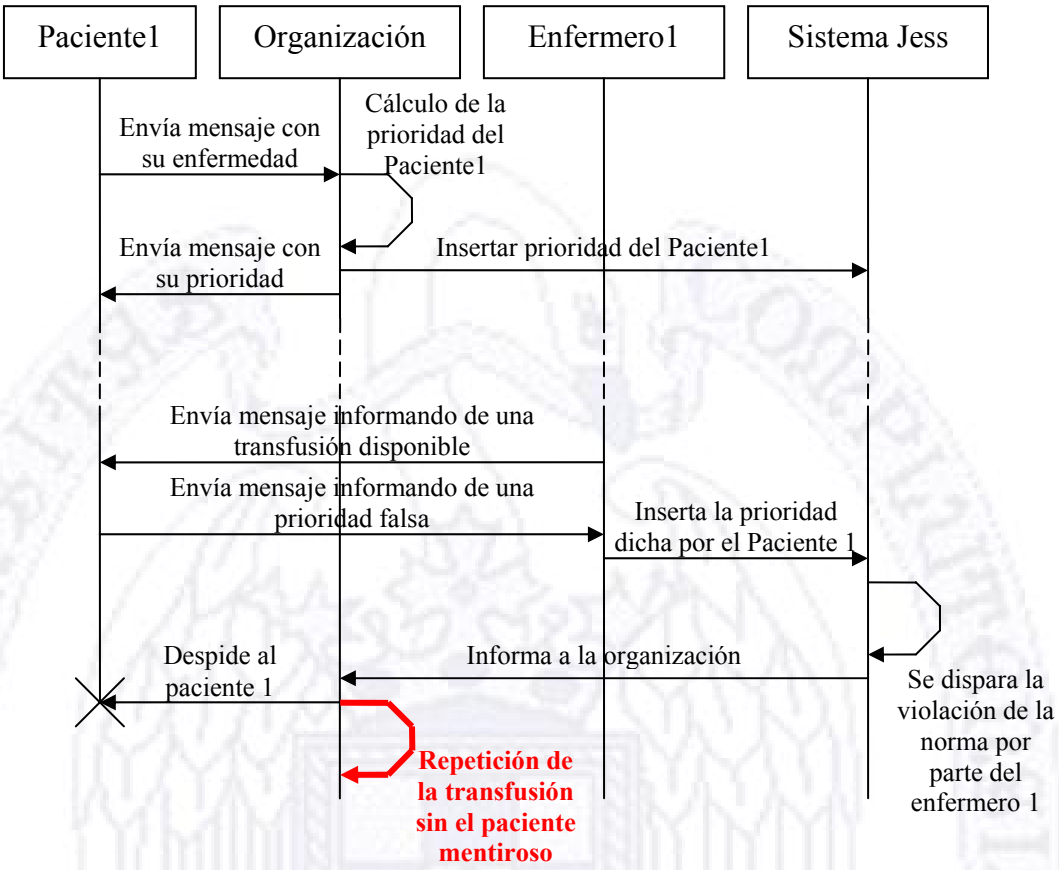
la organización de la ocurrencia de ese hecho. La organización busca en el entorno a ese enfermero y le elimina del sistema. En la siguiente figura se muestra el diagrama de secuencia de este escenario:



ii. **Un paciente viola una norma.** Si el enfermero informa de la prioridad de un paciente y la organización informó de otra prioridad distinta, el sistema Jess se da cuenta de que el agente ha violado la norma e informa a la organización de ese hecho. La organización busca en el entorno a ese paciente y le elimina del sistema. Aunque un agente haya mentido, se sigue con la ejecución de la transfusión y, si el paciente que tiene mayor prioridad no es el paciente que ha mentido, la transfusión se realiza a ese paciente de forma satisfactoria. En la siguiente figura se muestra el diagrama de secuencia de este escenario:



En el caso de que el paciente que ha mentido hubiera ganado la transfusión, se volvería a subastar la misma transfusión ya sin el paciente mentiroso. En la siguiente figura se muestra el diagrama de secuencia de este escenario:



4.6. Conclusión

La inclusión de normas en un sistema multi-agente se hace mucho más sencilla utilizando nuestra aplicación, puesto que el usuario no tiene que tener unos conocimientos avanzados de Jess.

Aún así, la falta de un Framework que adapte el sistema Jess a los MAS hace que el usuario tenga que conocer el funcionamiento de este sistema, puesto que tiene que realizar los asertos de las acciones y controlar los eventos que produce dicho sistema, que muchas veces tienen que ver con la implementación específica de las normas. Por ello, todavía puede resultar más cómodo a un usuario que posea ciertos conocimientos de Jess u otro sistema basado en reglas, la creación de una norma específica que genere los eventos con la información que precise para controlar el sistema.

Por estas razones, esta aplicación necesita ser utilizada por usuarios que tengan ciertos conocimientos de MAS, aunque complementada con otras herramientas que pueden ir apareciendo puede facilitar el auge de este tipo de sistemas, puesto que la inclusión de normas en ellos es indispensable.

La utilización de esta aplicación también puede estar justificada por parte de un usuario que posea ciertos conocimientos de Jess u otro sistema basado en reglas, puesto que le permite ahorrar bastante tiempo en la creación de cada regla de cada norma, e

incluso puede modificar la norma creada para que genere los eventos con la información que precise para controlar el sistema.

5. Conclusión

La primera motivación que puede causar trabajar en un proyecto de éste tipo es conocer cómo funcionan los agentes y cómo están formados los sistemas multi-agentes. Te permite descubrir lo que puedes hacer con ellos y a través de ellos. Hemos conocido cómo funcionan las normas y que al transformarlas a reglas escritas en Jess permiten controlar los comportamientos de los agentes.

Hemos intentado que a través de este trabajo se descubra y comprenda con facilidad los sistemas multi-agentes con la ayuda de las aplicaciones creadas durante el proceso; que con la aplicación sea más sencillo la creación de las normas y su transformación a las reglas descritas en Jess, creando así un sistema de reglas de manipulación de normas. Y que el ejemplo del sistema final pueda resolver el resto de dudas que pudieran surgir a lo largo de todo el trabajo.

Por lo tanto, la creación y ejecución de normas para sistemas multi-agente se simplifica de forma que el conocimiento necesario por el usuario es únicamente el de los conceptos básicos de cualquier aplicación compuesta por agentes.

Esperamos que nuestro trabajo pueda servir de ayuda en un futuro a otros investigadores. Y hemos conseguido cumplir los objetivos propuestos cuando elegimos el proyecto.

6. Apéndice

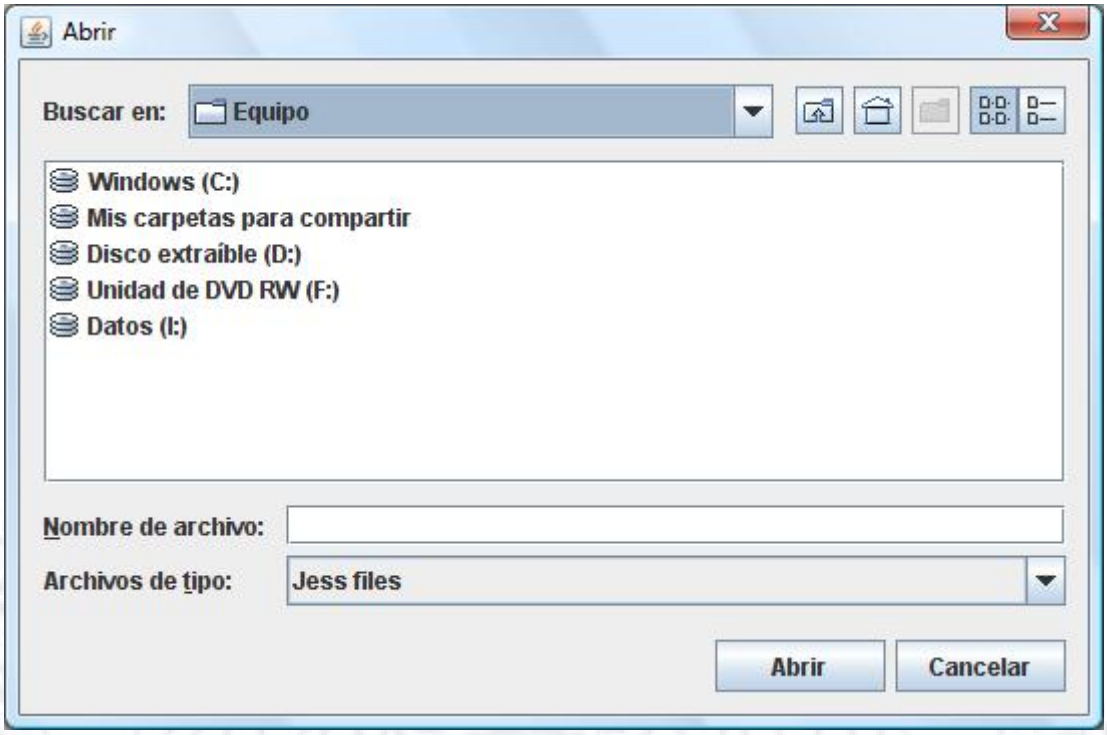
6.1. Manual de usuario del generador de normas

Ventana principal:

Acciones que podemos realizar:

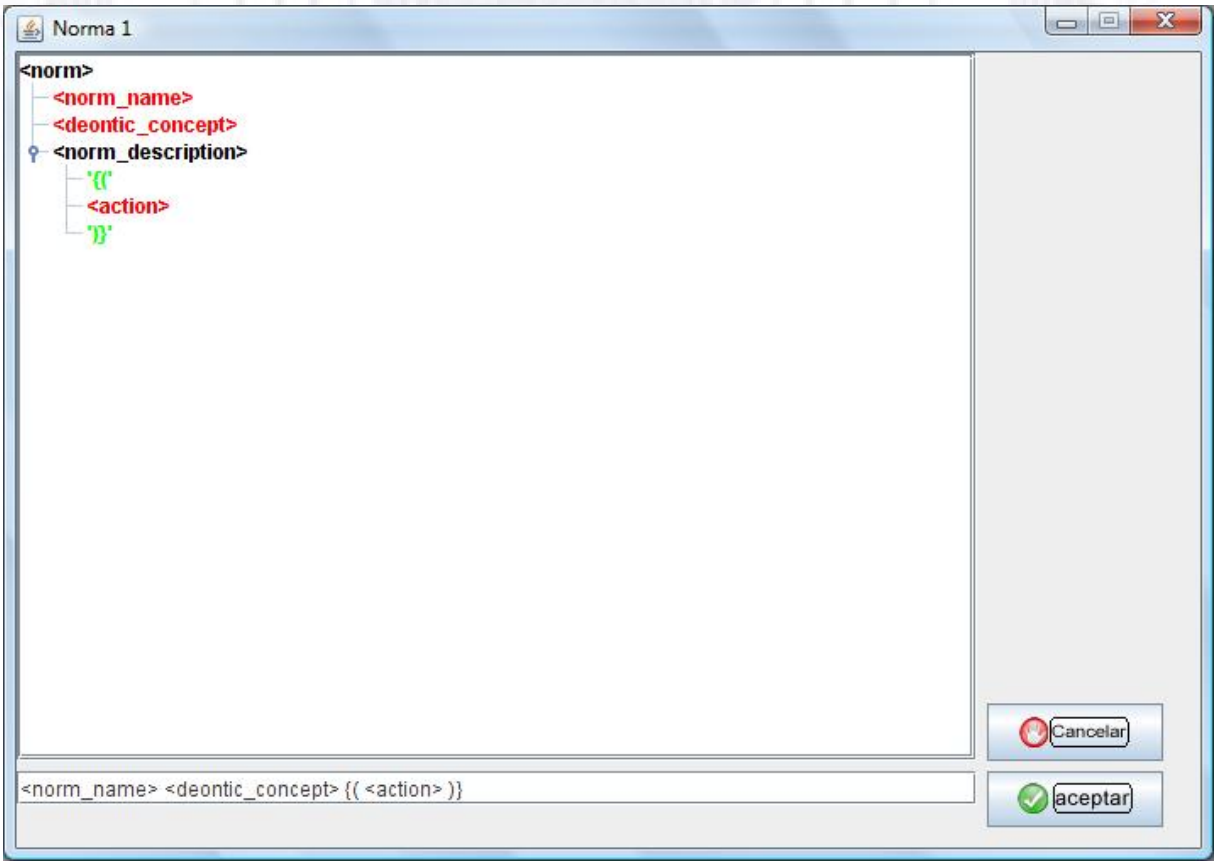
- Abrir una norma ya escrita: Para ello podemos pulsar el botón 

o pulsar Archivo > Cargar o el comando Ctrl-l. Aparecerá una ventana de selección de archivo como muestra la siguiente imagen.



- Añadir una nueva norma: pulsando el botón 

o pulsando Editar > Añadir norma o con el comando Ctrl-A.
Aparecerá una ventana nueva donde se podrá diseñar una norma desde el principio o escribirla directamente en el cuadro de texto habilitado para ello, que se encuentra en la parte inferior de la ventana.



Una vez escrita la norma, pulsando el botón aceptar, la norma se añadirá directamente a la ventana principal. Si se desean añadir más normas deberá repetir este paso las veces que se deseen.

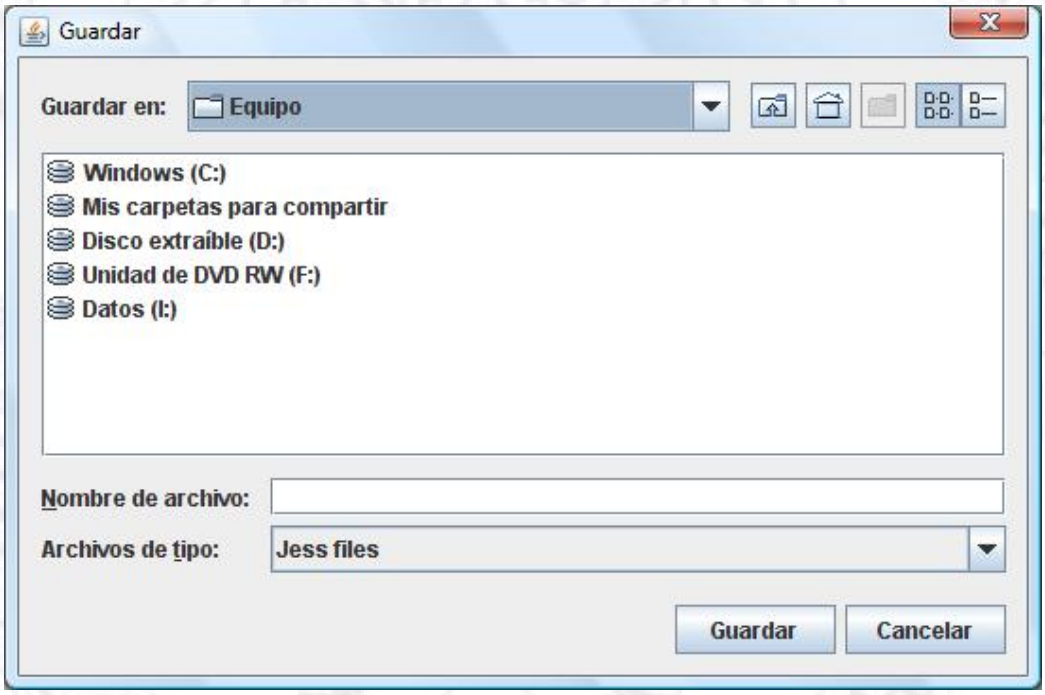
Para entender el funcionamiento de esta ventana consultar la subsección 6.1.1. que se encuentra al final de este punto.

- Borrar norma: Primero se debe seleccionar la norma en la ventana principal y a continuación se puede pulsar o bien el botón 

o Editar > Borrar norma o Ctrl-D.

- Guardar norma: Una vez que tengamos definidas todas las normas que debe cumplir nuestro sistema podemos guardar el fichero de reglas generadas a partir de dichas normas. Para ello podemos o pulsar el botón 

o Archivo > Guardar o Ctrl-S.
Aparecerá la ventana de selección de destino y nombre del archivo.



- Empezar desde el principio: Se puede o pulsar el botón 

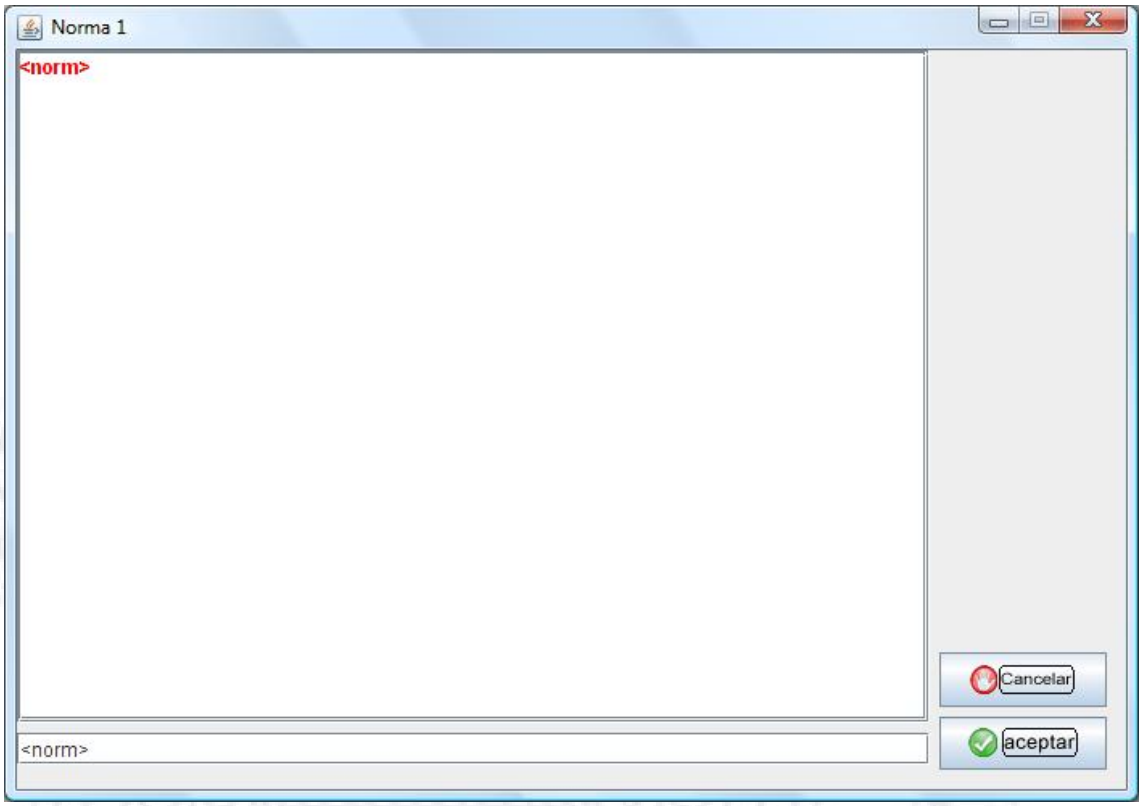
o con Archivo > Nuevo o con Ctrl-N.

- Salir del programa: Pulsando el botón 

o pulsando Archivo > Salir o con el comando Ctrl-C o pulsando X en la esquina superior derecha de la ventana.

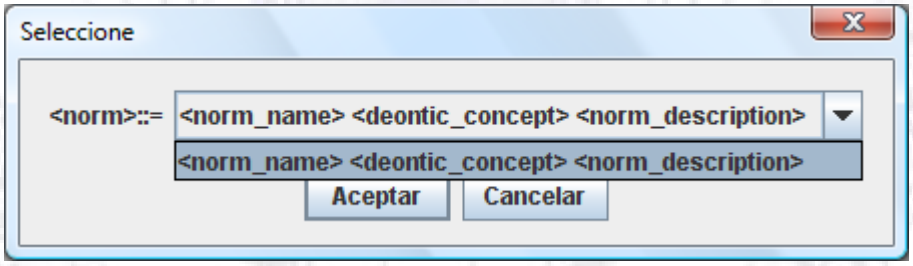
6.1.1. ¿Cómo crear una norma?

Inicialmente nos encontramos con la siguiente ventana:



Por defecto aparece la primera fase para la descripción de la norma <*norm*> aparecerá en color rojo, lo cual significa que aún no se ha desplegado, es decir que no se ha elegido una de sus opciones, para realizar esto debe pulsar sobre la palabra, que tiene como efecto la invocación de una nueva ventana.

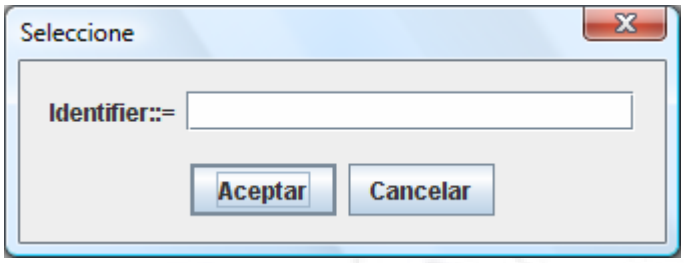
Dicha ventana es la de selección:



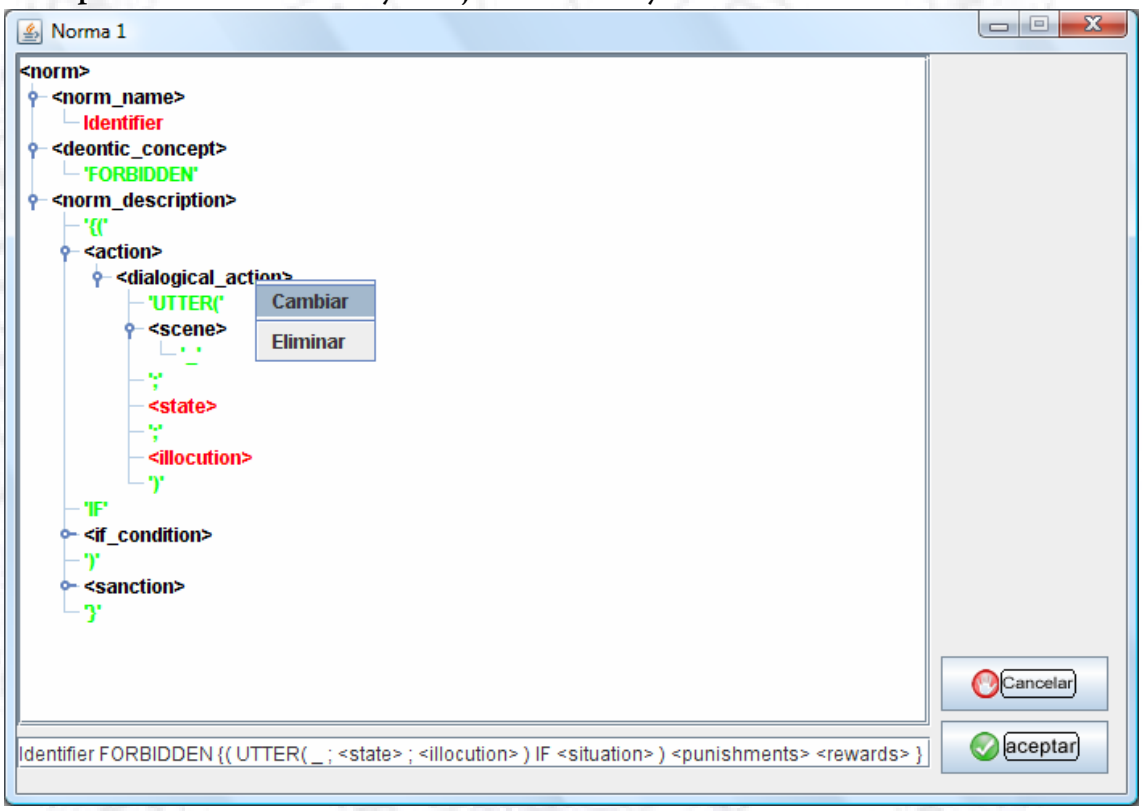
Como su propio nombre indica en ella podrás seleccionar la siguiente producción del no terminal.

Una vez elegida la siguiente producción y pulsado "aceptar" la ventana de selección se cerrará y consecuentemente el color del no terminal cambiará a negro.

Si en vez de aparecer una ventana de selección nos aparece una ventana de escritura eso quiere decir que hemos llegado a un terminal y debemos escribir el identificador y pulsar "aceptar" para introducirlo en el árbol.



Para deshacer una de nuestras selecciones es tan simple como pulsar con el botón derecho del ratón sobre el no terminal (en negro) en el cual nos habíamos equivocado o deseamos cambiar. Aparecerán dos selecciones o eliminar o cambiar. La primera acción provocará que todo lo que había por debajo de ese nodo en el árbol desaparecerá puesto que ahora el nodo pulsado se convertirá en hoja. La segunda opción elimina todo lo hay debajo de ese nodo y solicita el nuevo valor del nodo.



➢ Significado de los colores:

- Rojo: No terminal que aún no ha sido expandido.
- Verde: Terminal, no es posible seleccionarlo.
- Negro: No terminal, que ya fue expandido. Si se vuelve a pulsar con el botón izquierdo se desplegará o en caso contrario se contraerá.

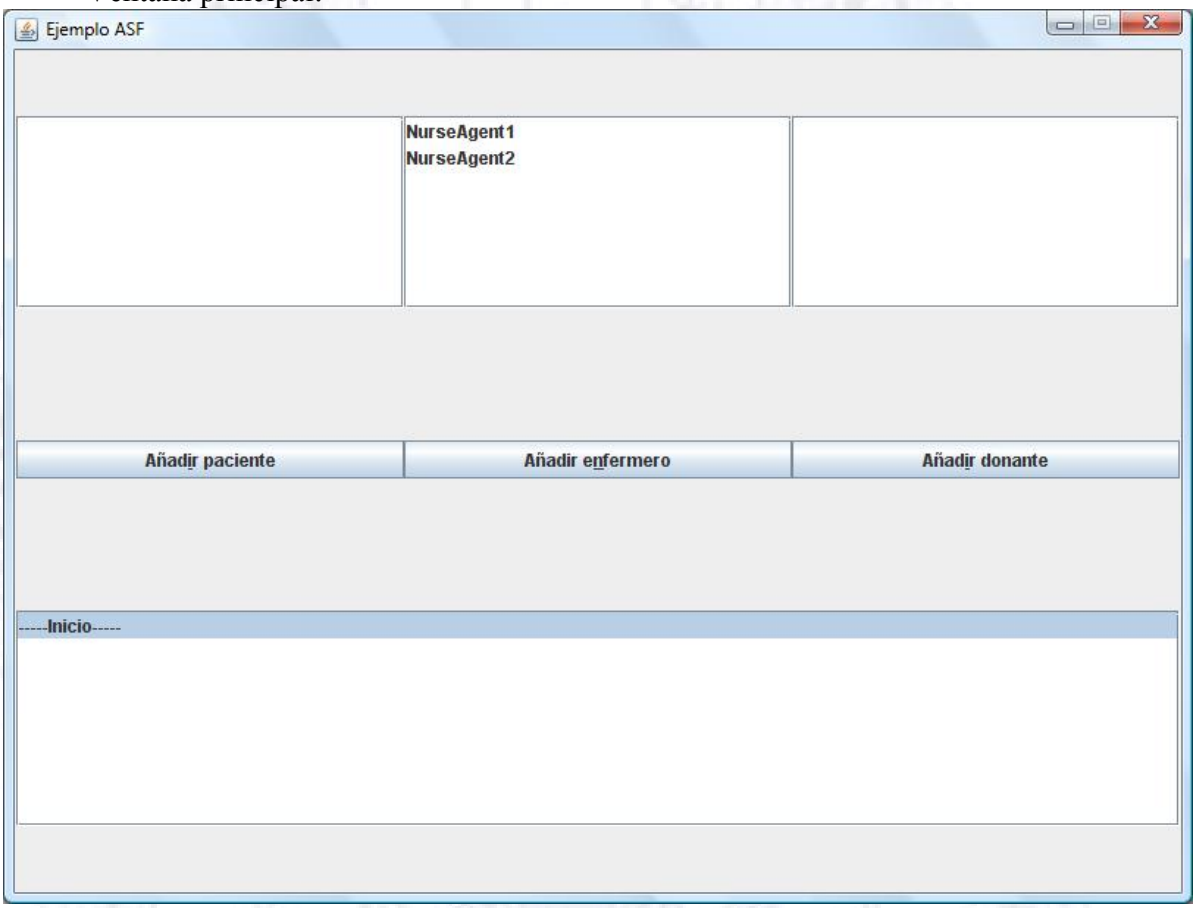
➢ Significado de las palabras:

- <palabra>: la palabra es un no terminal.
- 'palabra': la palabra es un terminal descrito en la norma.
- palabra (en color verde): es un terminal especificado por el usuario mediante un identificador.

6.2. Manual de usuario del ejemplo del sistema de donaciones y transfusiones de un hospital

Para ayudar a entender el funcionamiento del sistema hemos diseñado una aplicación, de forma que el usuario pueda interactuar con el sistema, como por ejemplo añadir pacientes, enfermeros o donantes y podrá ver la comunicación entre los distintos elementos que componen el sistema.

Ventana principal:



Podemos ver que la ventana principal está dividida horizontalmente en tres partes. La primera de ellas mostrará los agentes que hay dentro del hospital y está separado en tres bloques pacientes, enfermeros y donantes. La siguiente parte son botones con los que poder incluir nuevos elementos al hospital. Y la última parte es una pantalla donde se mostrará qué está ocurriendo en el hospital en cada momento y los mensajes enviados entre los elementos, para conseguir comunicarse.

Añadir paciente: Si pulsamos el botón con este nombre aparecerá la siguiente ventana:

Donde indicaremos el tipo de sangre y la enfermedad que padece. Cuando hayamos introducido todos los datos pulsamos el botón de Aceptar. Inmediatamente después el paciente aparecerá en la sección de pacientes de la ventana principal indicando su tipo de sangre y la prioridad que tiene para recibir una transfusión calculada en función de su enfermedad.

Añadir enfermero: Si pulsamos este botón se añadirá directamente un nuevo enfermero a la lista actual.

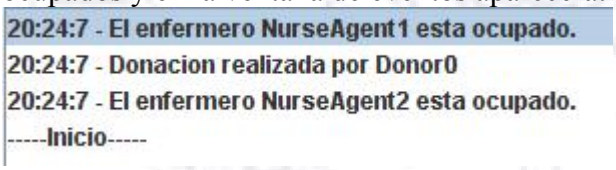
Añadir donante: Si pulsamos el botón de añadir donante saldrá la siguiente ventana:

Sólo tendremos que indicar el tipo de sangre que donará el donante.

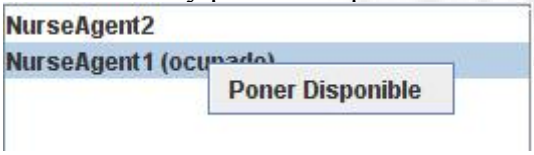
Supongamos que añadimos dos pacientes y un enfermero, antes de que empiecen las donaciones o transfusiones, siempre que sea posible que ocurran, tendremos:

Inicialmente en el sistema existen dos enfermeros, por lo tanto si hemos añadido uno en total tendremos tres enfermeros.

Si añadimos un donante, se realizará una donación, dos enfermeros pasarán a estar ocupados y en la ventana de eventos aparecerá:

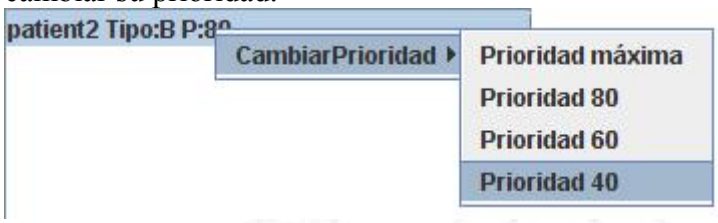


Después de la donación los enfermeros aparecerán como ocupados, para cambiar su estado a disponible solo es necesario seleccionarlo y pulsar con el botón derecho del ratón sobre él y poner a disponible.



Cuando ha entrado una donación al sistema, los enfermeros preguntan a los pacientes si son compatibles con el tipo de sangre y qué prioridad tienen.

Si seleccionamos un paciente y pulsamos con el botón derecho del ratón podemos cambiar su prioridad.



Cuando se encuentre una donación disponible para poder realizar una transfusión habrá un enfermero pregunte a los pacientes si son compatibles con la donación y su prioridad, el que tiene la prioridad cambiada mentirá. Si están cargadas las reglas en el sistema el paciente que ha mentido será eliminado. Y la transfusión se destinará al paciente que tenga mayor prioridad de los que no hayan mentido.

7. Referencias

1. Viviane Torres da Silva, “From the Specification to the Implementation of Norms”, Departamento de Sistemas Informáticos y Computación – UCM, Spain, Madrid, 2008

2. A. García-Camino, P. Noriega and J Rodríguez-Aguilar, “Implementing Norms in Electronic Institutions,” in Proceedings of AAMAS, ACM Press, 2005, pp. 667-673.

3. F. Lopiz y López, “Social Power and Norms: Impact on agent behavior,” PhD thesis, Univ. of Southampton, Faculty of Engineering and Applied Science, Department of Electronicsand Computer Science, 2003.

4. F. Lopiz y López, M. Luck and M. d'Inverno, “Constraining autonomy through norms,” in Proceedings of AAMAS, ACM Press, 2002 pp. 674-681.

5 J. Vázquez-Salceda, H. Aldewereld, and F. Dignum. Norms in Multiagent Systems: Some Implementation Guidelines. In 2nd European Workshop on Multi-Agent Systems, Barcelona, 2004

6 N. Jennings, “On agent-based software engineering”. Department of Electronics and Computer Science, University of Southampton, UK, 1999.

7 V. Torres da Silva, R. Choren y C. de Lucena “A UML Based Approach for Modeling and Implementing Multi-Agent Systems”. Computer Science Department, Pontificia Universidade Católica do Rio de Janeiro, Brasil, 2004.

8 V. Torres da Silva, M. Inés Cortés, C. de Lucena “An Object-Oriented Framework for Implementing Agent Societies”. 2004.

Los alumnos Manuel Féliz Díaz Jara, Inmaculada Magro García y Jorge Luís Queipo Bravo autores del proyecto de sistemas informáticos *Describiendo e implementando normas para sistemas multi-agente*, autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

En Madrid a 4 de Julio del 2008.

Fdo. Manuel Félix Díaz Jara

Fdo. Inmaculada Magro García

Fdo. Jorge Luis Queipo Bravo